Muchos procedimientos clásicos permiten al ordenador resolver algunos juegos sencillos. Otros más complejos, como el ajedrez, precisan técnicas más eficientes para su resolución.

La inteligencia artificial proporciona dichas técnicas. La aplicación de éstas hace que los programas se comporten de forma inteligente simulando el razonamiento humano.

El seguimiento de este libro y la realización de los ejercicios prácticos que en él se proponen introducirán al lector en el manejo adecuado de estos métodos.

Se estudian juegos tan conocidos como ajedrez, damas, dominó, mastermind, rompecabezas, torres de Hanoi, etc., además de los procedimientos básicos para enfrentarse con otros juegos o problemas similares.

7

Juegos inteligentes en microordenadores

esc

ENCICLOPEDIA PRACTICA DE LA

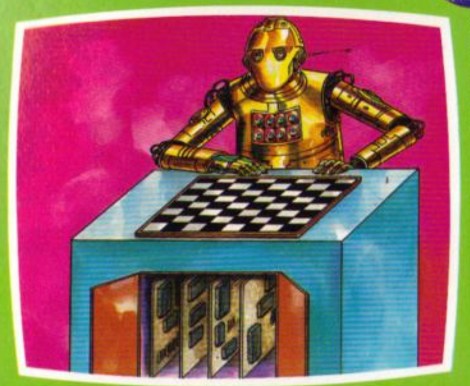
INFORMATICA APLICADA

7

Juegos inteligentes en microordenadores

GIA







Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA.

Tomo 7. **Juegos inteligentes en microordenadores**
Miembros del Laboratorio de Inteligencia Artificial de la FIM
FRANCISCO ASTUDILLO PACHECO, DANIEL BORRAJO MILLAN,
M.ª CARMEN TARRAT ALCALDE, IRMA TRUEBA VALLE

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Sor Angela de la Cruz, 24-7.º G. Teléf. 279 40 36. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28020 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-031-6
ISBN de la obra: 84-7688-018-9.

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-38.683-1986.
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Sor Angela de la Cruz, 24-7.º G. Teléf. 279 40 36. 28020 Madrid

Octubre, 1986.
P.V.P. Canarias: 365,-

INDICE

Los programas que aparecen en este libro funcionan en los ordenadores:

IBM-PC, XT, AT y compatibles.
AMSTRAD-464, 664, 6128, 1512.
SINCLAIR-SPECTRUM 48 K, 128 K, PLUS, PLUS 2.
MSX-Todos los modelos.
COMMODORE-CBM 64 y CBM 128.

PROLOGO

ES una satisfacción para cualquier persona que se dedique a la enseñanza el ver cómo un grupo de ex alumnos, y hoy colaboradores, se asoman al exterior mediante la publicación de libros, artículos en revistas, etc. Tras dos años de intenso trabajo en el laboratorio, durante el cual se han familiarizado con la Inteligencia Artificial y sus técnicas, sale a la luz este libro, que suple en parte la carencia de textos en idioma español que hay sobre este tema. Realmente, el trabajo realizado para escribir este libro es, además de meritorio, útil, pues incluye una gran parte de recopilación, síntesis y redacción, incorporando parte del conocimiento, tanto teórico como práctico, que poseen sobre la materia.

Es seguro que, tras leer este libro, el lector se va a sentir atraído por el área de la Inteligencia Artificial, lo que puede impulsar su utilización.

Por otra parte, el hecho de ser una obra de divulgación, en el sentido más noble del término, no le resta un ápice de rigor conceptual. Tanto las ideas como los ejercicios y ejemplos que las desarrollan son sencillos y adecuados para el objetivo propuesto.

Por todo ello, es posible afirmar que el éxito de la publicación, por merecido, está garantizado.

Juan Pazos Sierra
Vicedecano de Relaciones Exteriores
Facultad de Informática de Madrid

INTRODUCCION



LOS juegos han sido muy estudiados a lo largo de la historia e incluso se ha establecido un modelo matemático, desarrollando una serie de técnicas y algoritmos que resuelvan los juegos (que sepan jugar) conocido como teoría de juegos.

Los rompecabezas y juegos serán problemas fácilmente tratables por el computador en la medida en que sean precisos y estén bien formulados. Además, el conocimiento adquirido durante la resolución de los mismos puede ser transferido a problemas.

A la hora de resolver un juego aparece el probema de alcanzar una situación deseada. La situación inicial se irá modificando mediante movimientos o acciones que conduzcan al estado objetivo. Entre todas las acciones posibles habrá que elegir aquélla que sea más conveniente. La complejidad del problema radica en el elevado número de combinaciones existentes. En cada momento del juego debe considerarse el número de jugadas o acciones distintas que pueden realizarse, así como las futuras consecuencias de aplicación de cada una de ellas. La elección de una u otra acción influirá sobre las demás, por lo que el número de consideraciones a tener en cuenta en cada momento para asegurar la eficiencia de un movimiento puede llegar a ser inalcanzable, tanto para la mente humana como para la capacidad computacional de un computador. Dentro de este número tan elevado sólo unas pocas secuencias de acciones conducirán a la situación deseada.

En un principio, la teoría de juegos fue estudiada por la rama de la ciencia denominada Investigación Operativa (IO). Las técnicas que la IO proporciona para resolver un juego serán aplicables si existe un procedimiento suficientemente finito. Se entiende por suficientemente finito aquel algoritmo que se puede ejecutar utilizando una cantidad razonable de recursos de ordenador (tiempo y memoria).

Cuando no exista o no se conozca un algoritmo aplicable al juego o exista y se conozca pero no sea suficientemente finito, deberán aplicarse otras técnicas para la resolución del problema. Estas técnicas las proporciona la rama de la Informática denominada Inteligencia Artificial (IA).

Las técnicas aportadas por la IA son de gran utilidad, ya que reducen considerablemente el número de estados en la búsqueda de la solución; además, son fácilmente programables. Estos programas de IA con estrategias ganadoras llevan rápidamente hasta la solución final.

La IA como rama de la ciencia que estudia la resolución de problemas de forma inteligente está muy interesada en los juegos. Por ello, la IA ha sido muy criticada en los últimos años, ya que se ha concentrado excesivamente en la resolución de juegos y puzzles en lugar de tratar problemas de la vida real.

Los problemas de la vida real la mayoría de las veces no están suficientemente especificados, por lo que no pueden ser tratados de una forma adecuada por un computador. En cambio, los juegos poseen una serie de características que los convierten en problemas muy atractivos para ser resueltos por el computador. Algunas de estas características son: poseer un estado meta u objetivo, disponer de una serie de situaciones concretas iniciales prefijadas y contar con una serie de reglas específicas aplicables a lo largo del juego.

Una de las herramientas básicas de la IA es la utilización de métodos heurísticos, esto es, aplicación de reglas basadas en la experiencia a la resolución de problemas. Cada problema llevará a la aplicación de un determinado tipo de heurística, dependiendo de sus características.

Las heurísticas se introducirán en los métodos de búsqueda de soluciones, reduciendo considerablemente el número de caminos a tener en cuenta en el transcurso de una partida.

Otra de las herramientas básicas de la IA son los sistemas de producción. Se exponen unas nociones básicas sobre búsqueda heurística y sistemas de producción, así como un lenguaje adecuado para su programación, como es el LISP.

TECNICAS UTILIZADAS EN INTELIGENCIA ARTIFICIAL 1

ALGORITMOS DE BUSQUEDA



UANDO se plantea la posibilidad de escribir un programa que actúe de acuerdo con unas reglas determinadas en contra de un adversario, es decir, cuando se desea programar un juego, éste debe plantearse como un problema cualquiera al que ha de encontrarse solución, y no una solución cualquiera, sino la solución óptima.

La IA mostró desde sus comienzos un especial interés por los juegos, ya que en ellos el ser humano debe desarrollar y utilizar todas sus capacidades cognoscitivas e intelectuales, preocupación principal de los estudios en IA.

Así, pues, planteado el juego como un problema, pueden utilizarse para su resolución las técnicas de IA. Una de ellas es la búsqueda. El proceso de búsqueda, en general, es aplicable a todos los tipos de problemas en los que la adquisición y recuperación de información constituye una parte importante.

Existen muchos métodos de búsqueda que pueden asimismo clasificarse de formas muy diferentes atendiendo a criterios dispares. Todos ellos, sin embargo, dependen en gran medida de la situación a la que se aplican, es decir, se ven afectados por consideraciones como:

Conocimiento de la meta, es decir, de lo que se busca.

Existencia de solución.

Grado de reversibilidad.

Competitividad: esta característica alcanza gran relevancia en los juegos en los que intervienen dos jugadores, ya que es necesaria una estrategia distinta de la que se utilizaría en rompecabezas o juegos contra la naturaleza, es decir, sin adversario.

Medida del progreso hacia la meta.

Existencia de restricciones.



BUSQUEDA EN GRAFOS

Los grafos son sistemas de representación de información muy utilizados en IA, y muy adecuados a la representación de juegos, ya que permiten operar fácilmente con los estados o situaciones que se alcanzan en el transcurso de las jugadas, así como averiguar sus relaciones de dependencia, sus conexiones directas e indirectas, etc.

Existen múltiples variaciones de los procedimientos de búsqueda en un grafo, pero todos ellos pueden representarse de forma general mediante un procedimiento, que puede describirse informalmente como sigue:

1. Crear un grafo de búsqueda G, constando solamente del nodo inicial i. Colocar i en una lista demoninada ABIERTA.
2. Crear una lista denominada CERRADA, que inicialmente está vacía.
3. CICLO: si ABIERTA está vacía, salir con fracaso.
4. Seleccionar el primer nodo de ABIERTA, eliminarlo de ABIERTA y colocarlo en CERRADA. Llamar a este nodo n.
5. Si n es un nodo meta, salir con éxito, con la solución obtenida trazando un camino a i mediante los direccionadores en G.
6. Desarrollar el nodo n, generando el conjunto S de sus sucesores que no sean a su vez antecesores. Colocar estos miembros de S como sucesores de n en el grafo G.
7. Establecer desde cada uno de los miembros de S que no estén ya en G direccionadores a n; es decir, desde aquellos que no estén ni en ABIERTA ni en CERRADA. Añadir estos miembros de S a ABIERTA. Para cada miembro de S que ya estuviera en ABIERTA o CERRADA, decidir si se redireccionan o no los punteros dirigidos a n. Para cada miembro de S que ya estuviera en CERRADA, decidir, para cada uno de sus sucesores en G, si se redireccionan o no sus punteros.
8. Reordenar la lista ABIERTA, bien conforme a algún esquema arbitrario, bien conforme a algún mérito heurístico.
9. Ir a CICLO.

Este procedimiento tiene la generalidad suficiente como para abarcar una amplia variedad de algoritmos especiales de búsqueda en grafos. Genera un grafo explícito G denominado «grafo de búsqueda» y un subconjunto A de G denominado «árbol de búsqueda», que es el que viene determinado por los direccionadores o punteros que se instalan en el grafo.

El paso más importante del procedimiento es el paso número 8. En él se ordenan los nodos de la lista ABIERTA de tal manera que el «mejor» de ellos es el que se selecciona para desarrollarlo a continuación en el paso 4. De la elección correcta o incorrecta del orden de desarrollo de los nodos del grafo depende en gran medida la eficiencia del procedimiento. Por

ello, es necesario poner un especial cuidado en la ordenación que en este punto ha de realizarse.

Siempre que el nodo seleccionado para su expansión sea un nodo meta, el proceso termina con éxito. Entonces el camino que lleva a él desde el nodo inicial, es decir, la solución del problema (o la secuencia de jugadas o movimientos que han de hacerse para ganar en un juego), puede hallarse siguiendo la pista de los apuntadores o direccionadores en orden inverso.

El paso 7 es el encargado de asegurar la no repetición de caminos de búsqueda idénticos, que restarían efectividad y rapidez al procedimiento. Si el grafo implícito en que se busca es un árbol, se puede estar seguro de que ninguno de los sucesores generados en el paso 6 se generó previamente en otra iteración. Este es un caso especial en el que no sería necesaria la comprobación, ya que no existirían nodos sucesores de otro que pudieran ser a su vez antecesores del mismo.

Si el grafo sobre el que se realiza la búsqueda no es un árbol, es posible que algunos de los miembros del conjunto S ya hayan sido generados, es decir, estén ya en ABIERTA o en CERRADA. Si se continúa el algoritmo sin imponer la comprobación de si alguno de los nodos ya ha sido generado anteriormente, el árbol de búsqueda que se genera contiene varios nodos iguales, lo que aumenta el tamaño del grafo. Suele resultar más barato realizar la comprobación de la identidad de los nodos que examinar un enorme grafo con nodos repetidos. El procedimiento que se ha expuesto realiza esta comprobación, ya que sólo añade a la lista ABIERTA para su posterior expansión los nodos del conjunto S que no están ya instalados ni en la propia lista ABIERTA ni en la lista CERRADA.

La misión de los direccionadores o punteros es la de crear el árbol de búsqueda. Este debe conservar el camino de menor coste encontrado hasta el momento desde i a cualquier nodo. Cuando al expandir nuevos nodos se encuentra un camino de coste menor, se cambian los punteros para registrarlo como el nuevo camino de menor coste. El coste de un camino se calcula por medio de la suma de los costes asociados a cada uno de los arcos que lo forman. En los casos en que no existen costes asociados a los arcos, el coste de un arco se considera unitario.

Existe una variación del procedimiento que no genera todos los sucesores de un nodo a la vez. El procedimiento modificado genera sólo un sucesor cada vez, es decir, no coloca un nodo en la lista CERRADA hasta que se han generado todos sus sucesores. Esta modificación tiene gran utilidad en algunas aplicaciones específicas de IA.

BUSQUEDA EN PROFUNDIDAD

La técnica de búsqueda en profundidad, denominada también búsqueda lexicográfica canónica, consiste en dar prioridad a los nodos de niveles

más profundos en el grafo de búsqueda. Para ello es necesario asegurarse en cada paso de que cada nodo que se elige en el algoritmo traiga consigo la generación de todos sus nodos sucesores antes de proceder a la exploración de otro nodo.

Esta técnica es adecuada en los casos en que existen muchas soluciones y todas son igualmente deseables, o en casos en los que es fácil darse cuenta de que el camino que se está siguiendo no es correcto, ya que no conduce a la solución.

Una forma de implementación de este procedimiento a partir del procedimiento general ya expuesto es la de variar la estructura de la lista ABIERTA, cambiándola por una estructura de pila. Así, el procedimiento colocará los nuevos sucesores en la cima de ABIERTA, y seleccionará para su desarrollo el nodo generado más recientemente. Es decir, la reordenación de ABIERTA se realiza de forma implícita mediante su nueva estructura de pila. De esta forma se garantiza que se desarrollen, en primer lugar, los nodos con mayor profundidad.

La definición informal del procedimiento puede ser como sigue:

1. Definir el límite de nivel. Poner el nodo raíz en ABIERTA.
2. Si ABIERTA está vacía, salir con fracaso. En caso contrario, continuar.
3. Eliminar el nodo de la cima de la pila ABIERTA y pasarlo a CERRADA. Llamar a este nodo n.
4. Si el nivel del nodo n es igual al límite de nivel borrar CERRADA e ir al punto 2. En caso contrario, continuar.
5. Desarrollar n, generando todos sus sucesores. Colocar estos sucesores, sin ningún orden especial en la cima de ABIERTA y asignar a cada uno un direccionador hacia n.
6. Si cualquiera de estos sucesores es un nodo meta, terminar con la solución obtenida retrocediendo por medio de los punteros. En caso contrario, continuar.
7. Si cualquiera de estos sucesores es final sin éxito, eliminarlo de ABIERTA y eliminar CERRADA.
8. Ir a 2.

Por ejemplo, si se considera el arbol de búsqueda de la figura 1, el procedimiento de búsqueda en profundidad visitaría los nodos en el orden siguiente: A, B, E, F, I, C, D, G, J, K, H.

El procedimiento de búsqueda en profundidad resulta peligroso si no se controla adecuadamente en grafos grandes, especialmente en aquellos que presenten profundidad infinita. En estos casos, el procedimiento continuaría examinando nodos cada vez a una profundidad mayor sin tener posibilidad de volver atrás. El procedimiento puede resultar altamente ineficiente aun en grafos no infinitos, ya que puede seguir caminos de gran profundidad cuando la solución se encuentra mucho más cerca del comienzo del grafo. Pasa subsanar en lo posible estos problemas se utiliza un límite de nivel, que una vez alcanzado detiene el proceso de búsqueda evitando su prolongación infinita.

Este procedimiento es bastante económico en lo que se refiere a almacenamiento en memoria. La memoria máxima necesaria para su ejecución no excede del producto del límite de nivel por el factor de ramificación del grafo.

BUSQUEDA EN AMPLITUD

El procedimiento de búsqueda en amplitud recorre el grafo por niveles, examinando todos los nodos de un nivel antes de pasar al siguiente. Su estructura es similar a la del algoritmo de búsqueda en profundidad, sólo difieren en la ordenación que se hace de los nodos en ABIERTA, que en este caso tiene estructura de lista. La expresión del algoritmo es la siguiente:

1. Colocar el nodo inicial en ABIERTA.
2. Si ABIERTA está vacía, salir con fracaso. En caso contrario, continuar.
3. Eliminar el primer nodo de ABIERTA y pasarlo a CERRADA. Llamar a este nodo n.
4. Desarrollar n, generando todos sus sucesores. Asignar a cada uno un direccionador hacia n.
5. Añadir dichos sucesores al final de ABIERTA.
6. Si el primer nodo de ABIERTA es un nodo meta, terminar con la

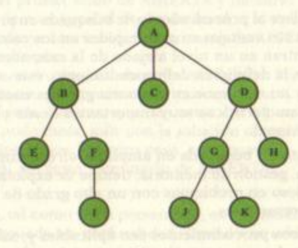


Fig. 1.

solución obtenida retrocediendo por medio de los punteros. En caso contrario, continuar.

7. Si el primer nodo de ABIERTA es final sin éxito, eliminarlo de ABIERTA y eliminar CERRADA.

8. Ir a 2.

Como puede observarse, añadiendo los sucesores de cada nodo al final de ABIERTA, van quedando por delante los nodos de menor profundidad, que son los que se van a desarrollar preferentemente.

Por ejemplo, si se considera el árbol de búsqueda de la figura 1, el procedimiento de búsqueda en amplitud visitaría los nodos en el orden siguiente: A, B, C, D, E, F, G, H, I, J, K.

La búsqueda en amplitud no presenta los problemas de la búsqueda de profundidad, por lo que puede utilizarse incluso en árboles y grafos de profundidad infinita. Encuentra la solución óptima al problema, entendiendo como tal la que se encuentra más cerca del punto de partida, que es, a su vez, la de mínimo coste.

Fácilmente puede deducirse que este procedimiento no resulta muy útil en los casos en que todos los caminos conducen al nodo final más o menos con el mismo nivel. Es un procedimiento bastante eficiente, aunque tiene la desventaja de necesitar una gran cantidad de almacenamiento en memoria.

Dada la importancia y amplia utilización de los dos procedimientos que se acaban de ver, amplitud y profundidad, han sido estudiados en conjunto y se ha contrastado su funcionamiento para situaciones similares. No existe un acuerdo total entre los distintos especialistas sobre las ventajas e inconvenientes de cada uno de los métodos, aunque se pueden extraer conclusiones bastante compartidas, que son las que se van a exponer a continuación.

En lo que se refiere al procedimiento de búsqueda en profundidad, pueden contarse entre sus ventajas su gran rapidez en los casos en que las soluciones se encuentran en un nivel alejado de la raíz; además, como puede comprobarse en la definición del procedimiento, éste coloca pocos nodos en ABIERTA, y no mantiene en memoria grandes cantidades de información a la vez, características muy importantes desde el punto de vista del coste computacional.

El procedimiento de búsqueda en amplitud ofrece importantes ventajas de optimización, gestión de memoria, tiempo de explotación, etc. Es especialmente ventajoso en problemas con un alto grado de irreversibilidad o prácticamente irreversibles.

En general, ambos procedimientos son aplicables y, salvo en casos extremos muy particulares, sus prestaciones resultan bastante similares.

BUSQUEDA CON VUELTA ATRAS

En gran parte de los problemas que se presentan en IA, y particularmente a la hora de abordar la programación de un juego, se plantean situaciones en las que ha de tomarse una decisión eligiendo sólo una entre varias alternativas. Esta elección puede conducir a la meta deseada o, por el contrario, puede conducir a un callejón sin salida o a un fracaso. En estos casos, por otra parte bastante frecuentes, el programa debe disponer de algún mecanismo que le permita volver sobre las elecciones realizadas y hacer otra elección diferente que pueda llevarle al objetivo buscado. Este proceso se denomina en programación «vuelta atrás» o «retroceso». También es bastante conocida y utilizada su denominación inglesa «backtracking».

Cuando se utiliza este procedimiento, cada vez que se llega a un punto muerto o no se consigue alcanzar la meta, se «desanda» el camino recorrido olvidando lo hecho desde la última toma de decisión, iniciándose de nuevo el proceso tomando otro camino no utilizado anteriormente. Claro está que el procedimiento resultará tanto más eficiente cuanto mayor acierto se tenga en la elección de las distintas alternativas. Por tanto, debe buscarse un método de clasificación u ordenación de las distintas posibilidades de desarrollo que permita seguir el camino más prometedor en cada ocasión.

Una formulación del algoritmo de vuelta atrás es la que se muestra a continuación:

1. Colocar el nodo inicial en ABIERTA.
2. Si ABIERTA está vacía, salir con fracaso. En caso contrario, continuar.
3. Examinar el primer nodo de ABIERTA y llamarlo n.
4. Si el nivel de n es igual al límite de nivel, o si todas las ramas que parten de n han sido ya atravesadas, eliminar n de ABIERTA e ir a 2. En caso contrario, continuar.
5. Generar un nuevo sucesor de n, llamándolo n'. Poner n' en la cima de ABIERTA y colocar un direccionador que apunte a n.
6. Marcar n para indicar que la rama (n n') se ha atravesado.
7. Si n' es un nodo meta, salir con la solución obtenida retrocediendo a través de direccionadores. En otro caso, continuar.
8. Si n' es final sin éxito, eliminarlo de ABIERTA.
9. Ir a 2.

Este algoritmo, tal como se ha presentado, es una modificación del procedimiento de búsqueda en profundidad. Cuando se selecciona un nodo para ser explorado sólo se genera uno de sus sucesores y éste se somete

de nuevo a exploración, a menos que sea el nodo meta o un final sin éxito. Si el nodo generado cumple algún criterio de parada, el programa retrocede al antecesor más cercano todavía no explorado, esto es, el que aún no ha generado sus sucesores.

En general, los criterios de parada que provocan la vuelta atrás son:

— La generación de un estado que ya hubiera aparecido anteriormente.

— La realización de un número previamente fijado de elecciones sin haberse alcanzado la situación objetivo.

— La imposibilidad de realizar otra nueva elección en el punto al que se ha regresado.

— La imposibilidad de llegar desde el estado actual al estado objetivo.

Este procedimiento consigue un mayor ahorro de memoria que el procedimiento de profundidad, ya que no necesita almacenar más que el camino que está siguiendo en ese momento, despreciando el resto del árbol de búsqueda.

Existen variaciones más sofisticadas del algoritmo de vuelta atrás que permiten realizar el retroceso no al nivel anterior, sino a un punto situado varios niveles más arriba. Este procedimiento se denomina «vuelta atrás por niveles».

PROCEDIMIENTO DE GENERACION Y VERIFICACION

Para ciertos problemas de características particulares los procedimientos de búsqueda que ya se han expuesto no resultan adecuados. A veces es conveniente realizar la búsqueda dividiendo el proceso en dos partes. La primera consiste en la generación de posibles soluciones, en el caso de un juego serían posibles situaciones vencedoras; y la segunda consiste en un mecanismo de verificación de dichas soluciones. Como puede observarse, esta estrategia es de una gran simplicidad, sin embargo, no es excesivamente eficiente en un caso general.

La formulación del procedimiento puede realizarse de la siguiente forma:

1. Generar una posible solución al problema planteado.
2. Verificar que esta solución es verdaderamente una solución al problema. La verificación suele hacerse generalmente por comparación.
3. Si la solución es válida, abandonar la búsqueda. En caso contrario, ir a 1.

El procedimiento encuentra la solución siempre que ésta exista y que la generación de soluciones se realice de forma sistemática. El grave inconveniente de este procedimiento es su gran consumo de tiempo, especialmente si el espacio problema (situaciones posibles) es grande. Si se combina este procedimiento con otras técnicas que restrinjan el espacio en el que ha de efectuarse la búsqueda, puede resultar muy efectivo. Una técnica adecuada para realizar esta combinación es la planificación, que ya ha sido probada con excelentes resultados.

PROCEDIMIENTO DE ESCALADA

El procedimiento de búsqueda por escalada puede considerarse como una variante de procedimiento de generación y verificación. Si la función de verificación se incrementa con una función que proporcione una estimación de la proximidad de un estado cualquiera al estado meta, el procedimiento se convierte en uno de escalada. Con este procedimiento se consigue reducir el número de secuencias de estados que hay que seguir antes de encontrar un estado final haciendo más rápido el método.

En esencia, el procedimiento de escalada se basa en la utilización de una función de evaluación sobre todos los estados que han de representarse en la búsqueda, incluidos los estados inicial y meta. La función ha de ser de tal forma que alcance su mayor valor para el estado final y no supere este valor máximo en ninguno de los estados intermedios. Cada problema concreto requiere una función de escalada diferente, que el programador ha de elegir teniendo en cuenta las características del problema que se trate.

El procedimiento, partiendo del estado inicial, elige entre todos los posibles aquel estado al que puede llegarse en un solo paso desde el inicial y que tiene un valor mayor de la función de evaluación elegida. De esta forma el procedimiento «escala» hacia la meta por el camino de máxima pendiente. En caso de que existan dos estados con el mismo valor de la función y éste sea máximo, debe elegirse uno de ellos, aun a riesgo de no elegir el correcto. En determinados problemas puede ser de ayuda disponer de cierta información adicional que pueda determinar cuál de los dos caminos es el más adecuado.

En el ejemplo de la figura 2 puede observarse el camino seguido hasta llegar a la solución 2, nodo J. Como ya se ha explicado, el procedimiento elige en cada caso el nodo cuyo valor de la función de evaluación f es mayor.

Este método plantea algunos problemas y, en determinados casos, no alcanza el estado final en el primer intento. Aun así, todavía puede seguir utilizándose la escalada, aunque realizándose en dos pasos. Esto es, eligiendo la secuencia de dos pasos de acciones a partir del nodo que haya tenido el valor máximo de la función de escalada. Esta modificación resuelve el problema en la mayoría de las ocasiones.

Cuando el método de escalada fracasa, suele ser debido principalmente a una elección equivocada de la función de evaluación. Esta, en general, es una función unidimensional, pero hay problemas en los que la meta difiere del estado inicial en varias dimensiones significativas. En estos casos resulta más adecuada una función de escalada multidimensional. En otros casos es posible guiar el procedimiento en base a la proximidad a la meta evaluada sobre cada una de las dimensiones por separado. Sin embargo, puede no ser fácil encontrar una manera sencilla de combinar las operaciones sobre cada dimensión en una función de evaluación simple que las englobe. Si se produce este caso ha de utilizarse un vector de evaluación asociado a cada estado, en lugar de un valor numérico único.

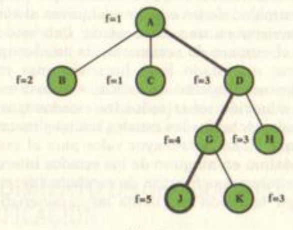


Fig. 2.

La estrategia de escalada goza de una gran popularidad entre los métodos de búsqueda, principalmente debido al escaso esfuerzo computacional que requiere su implementación. El ahorro de esfuerzo viene dado por el hecho de que el procedimiento no tiene necesidad de memorizar los intentos pasados ni el camino que conduce a la situación actual.

Evidentemente, el procedimiento presenta algunos fallos. Es fácil y ocurre con una cierta frecuencia, que la escalada quebrante el requisito de sistematicidad de la búsqueda, dejando algunos nodos sin visitar, es decir, estados posibles sin evaluar. La probabilidad de que se produzca este error puede disminuirse con la utilización de una función de escalada muy informativa respecto al problema que se trata. Hay que tener en cuenta, sin embargo, que la mejora en la función de evaluación puede conducir a una búsqueda a más profundidad de caminos que no llevan a la solución.

Así, pues, la escalada es una técnica útil en los casos en que se posee una función de evaluación altamente informativa que conduzca rápidamente hacia la cumbre global en la que se encuentra el estado meta. Asimismo es útil cuando los operadores que se aplican para pasar de un estado a otro son independientes, es decir, la utilización de un operador no impide la aplicabilidad futura de los otros operadores.



EL ALGORITMO A

Este algoritmo es una particularización del procedimiento general de búsqueda en grafos. La particularización consiste en la forma especial en que se reordenan los nodos en la lista ABIERTA. Para la ordenación de esta lista se utiliza una función de evaluación f(.) cuya misión es la de servir de estimador de la suma del camino de coste mínimo desde el nodo inicial al que se está considerando y el coste de un camino de coste mínimo desde el nodo considerado a un nodo meta.

De acuerdo a la definición de esta función, se decide ordenar los nodos de la lista ABIERTA de menor a mayor valor de la función de evaluación. De esta forma se expanden en primer lugar aquellos nodos que parecen más prometedores, es decir, los que forman un camino de coste mínimo.

Evidentemente, en la utilización del algoritmo sobre un grafo concreto no es posible utilizar una función de evaluación f(.) exacta que incorpore en su cálculo el coste de un camino de coste mínimo desde un nodo cualquiera a un nodo meta. Si así fuera, se conocería el mejor camino hasta la meta de antemano, situación en la que no sería necesario ningún tipo de búsqueda. Debido a ello se utiliza una función de evaluación f(.) aproximada al valor real. Podemos considerar esta función de la siguiente forma:

$$f(.) = g(.) + h(.)$$

donde:

— g(.) es el coste del camino en el árbol de búsqueda creado por el algoritmo desde el nodo inicial al nodo considerado. Este coste se calcula como la suma de los costes asignados a cada uno de los arcos. En los casos en que no existe tal coste, se supone unitario.

— h(.) es una función heurística, estimador del camino de coste mínimo desde el nodo considerado al nodo meta. Esta función depende de las características de cada problema, no existiendo un patrón de construcción de funciones de este tipo.

La eficacia del procedimiento depende en gran medida del acierto en la elección de esta función heurística h.

Si la función h(.) cumple la condición de mantenerse menor o igual que el coste real desde el nodo a la meta para cada uno de los nodos de la búsqueda, el algoritmo se denomina A*. Esta condición permite asegurar la admisibilidad del algoritmo, es decir, que la solución encontrada por el algoritmo es la óptima.

Existen multitud de fomulaciones teóricas sobre las propiedades y características del algoritmo, expresadas la mayoría en forma de teoremas.



BUSQUEDA EN GRAFOS Y/O

En determinados tipos de problemas existen relaciones de dependencias especiales que no pueden representarse en los grafos que se han estudiado hasta ahora. Estos problemas hacen uso de un tipo particular de grafo, denominado Y/O, o también grafo conjuntivo/disyuntivo.

Un grafo de este tipo consta de nodos etiquetados. Cada uno puede tener etiqueta Y o etiqueta O. Estas etiquetas se utilizan para indicar la relación existente entre un nodo y sus sucesores. Si un nodo tiene etiqueta Y no puede ser resuelto, a menos que lo sean también exactamente todos los sucesores. Si, por el contrario, está etiquetado O, puede ser resuelto si se resuelve alguno de sus sucesores.

La figura 3 muestra un ejemplo de un grafo Y/O. Como puede observarse, en las representaciones gráficas se utilizá la convención de no colocar etiquetas en cada nodo, representando los nodos Y mediante un arco que une las ramas que parten de él. Los nodos O no tienen esta marca.

Los algoritmos de búsqueda para grafos de este tipo no difieren mucho de los utilizados para los grafos simples. La diferencia estriba casi esencialmente en la determinación de las condiciones de finalización de la búsqueda. Mientras que la condición de terminación para grafos normales depende sólo de las propiedades de un nodo, en los grafos Y/O la condición

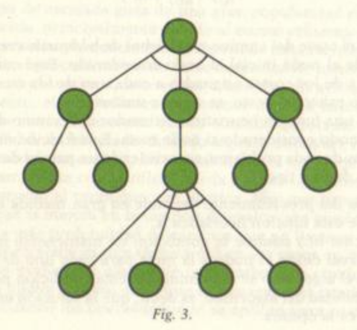


Fig. 3.

ha de buscarse sobre un conjunto de nodos teniendo en cuenta sus relaciones dentro del grafo.

Se puede considerar que existen los mismos procedimientos de búsqueda para los dos tipos de grafos, tales como amplitud, profundidad, retroceso, etc. Sin embargo, existe un algoritmo específico para los grafos Y/O, denominado algoritmo AO*. Este algoritmo, cuya estructura es prácticamente igual a la del algoritmo A*, utiliza también una función de evaluación que incluye una función heurística dependiente del problema al que se destine. Incorpora una función de purga que elimina del árbol de búsqueda aquellos nodos que no van a afectar al camino hasta entonces obtenido. Esta purga puede resultar muy útil, pero su uso ha de ser cauteloso, ya que en determinados casos da lugar al abandono de caminos que deberían haber sido explorados.



ESTRATEGIAS ESPECIALES PARA JUEGOS DE DOS CONTRINCANTES

Hasta el momento se han estudiado estrategias y procedimientos de búsqueda habituales en la resolución de problemas de Inteligencia Artificial. Todos ellos son utilizables en la programación de cualquier juego que pueda realizar una persona. De hecho, muchos de los programas de juegos que se presentan en este libro utilizan algoritmos de búsqueda como los ya mencionados. Suelen ser los juegos que se denominan «contra la naturaleza», es decir, sin adversario. Sin embargo, para los tipos de juego en los que intervienen dos jugadores, parece que los algoritmos de búsqueda no dan un resultado demasiado bueno. Esto se debe principalmente a su enorme cantidad de posibilidades, que hacen prácticamente imposible la generación y examen exhaustivo de todo el árbol de estados. Esto ocurre en juegos tan comunes como las damas o el ajedrez. Así, pues, el estudio profundo de los juegos de este tipo condujo al desarrollo de nuevas estrategias de búsqueda más eficientes en los casos mencionados que los algoritmos de búsqueda convencionales.

Estos algoritmos explotan las características de los juegos entre adversarios y perfectamente informados. Esto quiere decir que existen dos jugadores que alternan sus movimientos y turnos, y que en cada instante las reglas del juego definen qué movimientos se permiten y cuál será su efecto inmediato. Por tanto, no se consideran todos aquellos juegos en los que intervienen el capricho del azar, como pueden ser la mayor parte de los juegos de cartas. La denominación de perfectamente informados se debe al hecho de que cada jugador tiene en todo momento información total sobre la situación en que se encuentra su oponente y sobre las elecciones a las que puede optar él mismo.

Para la representación de los posibles estados del juego se utiliza la representación en forma de grafo, pero en este caso el grafo adquiere forma de árbol. Este árbol se construye a partir del nodo inicial o raíz, que representa la situación antes de comenzar el juego. Los sucesores de la raíz son las posiciones que puede alcanzar el primer jugador en un movimiento. A partir de cada uno de ellos se generan sus sucesores, que corresponden a las situaciones producidas por las respuestas posibles del segundo jugador. Los nodos terminales, también denominados hojas, representan las situaciones finales del juego y pueden etiquetarse con el resultado obtenido, es decir, GANADOR, PERDEDOR o EMPATE. Si se sigue el camino que lleva desde el nodo inicial a uno de los finales se obtendrá el desarrollo completo de una partida de juego, incluyendo el resultado, que viene dado por la etiqueta del nodo terminal.

De acuerdo con esta representación, y para facilitar la exposición de los algoritmos, se puede convenir en denominar al primer jugador MAX y al segundo MIN. Así, es posible referirse al turno del MAX o a la posición del MIN, entendiendo con ello las que corresponden al primer y segundo jugador, respectivamente. Por supuesto, los árboles, que representan los juegos contienen dos tipos de nodos: nodos MAX y nodos MIN. La figura 4 muestra un árbol de este tipo.

La resolución de un árbol con esta estructura consiste en el etiquetado de nodo raíz con una de las situaciones finales posibles, GANADOR, PERDEDOR o EMPATE. Para cada etiqueta de la raíz existe una estrategia óptima de juego que asegura esa situación final independientemente de la forma en que juegue el contrario. Las estrategias pueden definirse en función de subárboles del árbol original del juego, pudiendo definirse estrategias para los dos jugadores, es decir, tanto para MAX como para MIN. De todas las estrategias definibles tienen especial interés las denominadas «estrategias ganadoras», que aseguran la etiqueta GANADOR para la raíz independientemente de cómo juegue el oponente.

La teoría general existente sobre árboles y grafos aplicada a este caso conduce a resultados teóricos muy interesantes y de gran importancia, pero que rebasan las pretensiones de este libro.

Aunque esta forma de representación resulta muy adecuada y compacta, parece obvio que en la mayoría de lo juegos es demasiado larga la generación completa del árbol con el objeto de llegar a etiquetar sus nodos terminales y evaluar luego los nodos de forma ascendente para decidir en cada paso el movimiento óptimo, que es el que sigue el camino hallado. Por ejemplo, un árbol completo para el juego de las damas puede tener sobre unos 10^{40} nodos no terminales, cifra prácticamente imposible de alcanzar, ni siquiera con la capacidad de un ordenador moderno.

Evidentemente, resulta absurdo pretender una generación exhaustiva del árbol del juego. Incluso si se conociera una estrategia de juego en términos de las respuestas adecuadas para cada movimiento del contrario, el

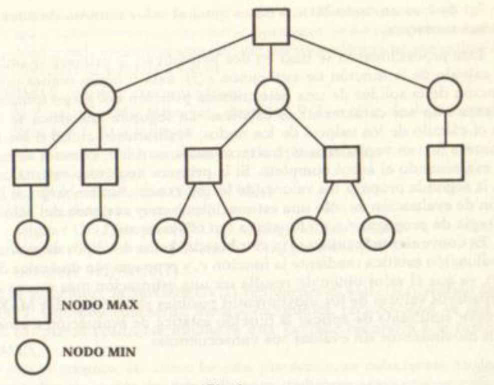


Fig. 4.

número de nodos que debería contener el árbol sería del orden de la raíz cuadrada del número total de nodos. Para el caso visto anteriormente, la cifra ascendería a 10^{20}, todavía intratable. Por tanto, debe abandonarse la idea de encontrar o retener una estrategia que ofrezca garantía absoluta del éxito del juego.

Las estrategias que trabajan con árboles de juegos no consideran el árbol completo. Sólo extienden el árbol hasta un determinado nivel. Los nodos de este nivel, que en el árbol completo serían los nodos terminales, se denominan nodos frontera de la búsqueda.

Existen diferentes estrategias de búsqueda en un árbol de juego, pero la mayoría se basan en la utilización de funciones de evaluación de las posibilidades de cada movimiento (análogas a las funciones de evaluación de los algoritmos de búsqueda anteriormente estudiados), y en la aplicación de la Regla Minimax. Esta regla consiste en un procedimiento de asignación de valor a los nodos de un árbol. La formulación del procedimiento es la siguiente:

1. Si el nodo J está situado en la frontera de búsqueda, su valor V(J) es igual al valor de la función de evaluación para dicho nodo, es decir, V(J) = e(J); de lo contrario:

2. Si J es un nodo MAX, V(J) es igual al valor máximo de cualquiera de sus sucesores.

3. Si J es un nodo MIN, V(J) es igual al valor mínimo de cualquiera de sus sucesores.

Este procedimiento se basa en dos heurísticas; la primera se utiliza en el cálculo de la función de evaluación e(J). Esta función realiza una estimación de la solidez de una determinada posición del juego tomando en cuenta sólo sus características estáticas. La segunda heurística se utiliza en el cálculo de los valores de los nodos, realizándolo como si los nodos frontera fueran verdaderos nodos terminales, es decir, como si se estuviera examinando el árbol completo. Si la primera heurística resulta correcta, la segunda propaga los valores de forma exacta. Sin embargo, si la función de evaluación es sólo una estimación no muy acertada del valor real, la regla de propagación no funciona tan eficazmente.

Es conveniente la utilización combinada de los dos tipos de evaluación (evaluación estática mediante la función e, y propagación dinámica del valor), ya que el valor obtenido resulta ser una estimación más exacta de los verdaderos valores de los movimientos posibles para el jugador MAX, que el valor resultante de aplicar la función estática de evaluación a esos mismos movimientos sin evaluar sus consecuencias.

ESTRATEGIA MINIMAX

La estrategia minimax es una estrategia de búsqueda exhaustiva. Su ventaja frente a una estrategia de búsqueda de las que ya se han visto es que no estudia el árbol completo. Elige siempre, en una situación determinada, la «mejor primera jugada», esperando la respuesta del adversario y volviendo a decidir en consecuencia.

El procedimiento define el estado de un nodo J y su valor minimax V(J) mediante un proceso de inducción descendente que puede transformarse de forma sencilla en un procedimiento de búsqueda recursivo. Una posible formulación del algoritmo puede ser:

MINIMAX (versión en profundidad)

1. Si J es un nodo terminal, devolver V(J) = e(J); de lo contrario:
2. Generar los sucesores de J, J_1, J_2, ...,J_b.
3. Evaluar $V(J_1)$, $V(J_2)$,..., $V(J_b)$ de izquierda a derecha.
4. Si J es un nodo MAX, devolver $V(J) = \max [V(J_1),..., V(J_b)]$.
5. Si J es un nodo MIN, devolver $V(J) = \min [V(J_1),..., V(J_b)]$.

Este procedimiento recorre el árbol de búsqueda de izquierda a derecha en forma de algoritmo de búsqueda en profundidad. El gasto de almacenamiento en memoria no es excesivo, ya que pueden generarse y evaluarse los sucesores de cada nodo de uno en uno y actualizar el nodo padre cada vez que se evalúa uno de sus nodos hijo. Esta forma de evaluación es más económica que la generación de todos los sucesores a la vez, que obliga a mantenerlos en memoria hasta que se ha obtenido su valor.

También puede escribirse una versión del procedimiento que utilice retroceso. Tendría la siguiente forma:

MINIMAX (versión con retroceso)

1. Si J es un nodo terminal, devolver V(J) = e(J); de lo contrario:
2. Para k = 1, 2,..., b hacer:
   a. Generar el k-ésimo sucesor de J, llamarlo J_k.
   b. Evaluar $V(J_k)$.
   c. Si k = 1, asignar $CV(J) \leftarrow V(J_1)$
   De lo contrario, para $k \geq 2$:
   asignar $CV(J) \leftarrow \max [CV(J), ..., V(J_k)]$ si J es MAX o
   asignar $CV(J) \leftarrow \min [CV(J), ..., V(J_k)]$ si J es MIN.
3. Devolver V(J) = CV(J).

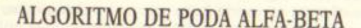
En el algoritmo, la variable CV(J) representa el valor actualizado para el nodo J. Este procedimiento incorpora también recursividad, ya que el paso 2 (b) puede realizarse mediante una llamada recursiva a la función MINIMAX.

El procedimiento, tal como ha sido planteado, es exhaustivo, es decir explora todos los nodos del árbol. Pueden realizarse versiones no exhaustivas, que pasen por alto los nodos que no proporcionan información útil. Una forma de ver esto es la formulación del procedimiento de resolución de un árbol de juego muy sencillo en el que los nodos terminales sólo tienen dos valores posibles, es decir, e(J) puede tomar sólo uno de los valores GANADOR o PERDEDOR. El procedimiento se denomina RESOLVER.

RESOLVER

1. Si J es un nodo terminal, devolver su estado directamente; de lo contrario:
2. Comenzar la resolución de los sucesores de J de izquierda a derecha (llamando a RESOLVER recursivamente).
3. Si J es un nodo MAX, devolver GANADOR tan pronto como se encuentre un sucesor que sea GANADOR; devolver PERDEDOR si todos los sucesores de J son PERDEDOR.
4. Si J es un nodo MIN, devolver PERDEDOR tan pronto como se encuentre un sucesor que sea PERDEDOR; devolver GANADOR si todos los sucesores de J son PERDEDOR.
5. Si J es un nodo MIN, devolver PERDEDOR tan pronto como se encuentre un sucesor que sea PERDEDOR; devolver GANADOR si todos los sucesores de J son GANADOR.

Este procedimiento puede implementarse, con ligeras modificaciones, para juegos con valores múltiples de la función de evaluación estática o incluso para valores continuos de la misma.



ALGORITMO DE PODA ALFA-BETA

El algoritmo alfa-beta es el más utilizado en las aplicaciones referidas a juegos, dada su excepcional utilidad en el aumento de la velocidad de la búsqueda sin producir pérdidas de información. El algoritmo determina el valor minimax de la raíz en el árbol del juego, recorriéndolo en orden predeterminado, por ejemplo, de izquierda a derecha, y desechando los nodos que no puedan afectar al valor minimax de la raíz. El nombre de «poda» se debe precisamente a que deja nodos sin examinar, cortando caminos que no aportan ninguna información útil. De esta forma también se aumenta considerablemente la velocidad de la búsqueda, evitando la expansión y evaluación de nodos no útiles.

El esquema de la poda puede resumirse de la siguiente forma:

Realizar la versión de MINIMAX con retroceso con una excepción; si en el proceso de actualización del valor minimax de un nodo este valor pasa de un cierto límite prefijado, no es necesario continuar la exploración por debajo de ese nodo. Su valor almacenado en la variable CV puede entonces transmitirse a su nodo padre como si ya se hubieran evaluado todos sus hijos.

Como puede verse, el algoritmo establece un valor frontera, a partir del cual considera que el valor de los nodos sucesores no afecta al valor que se va a obtener para el nodo padre. De esta forma puede olvidar todos los nodos que parten desde el que se está evaluando hacia abajo. El algoritmo considera dos valores límite, denominados alfa (α) y beta (β), uno inferior y otro superior. Estos valores de corte se ajustan de forma dinámica a medida que avanza la ejecución del algoritmo y se transmiten de arriba a abajo como sigue:

1. Límite α: el valor de corte para un nodo MIN es un límite inferior, denominado alfa, igual al valor más alto de los antecesores MAX de dicho nodo. En el momento en que el valor de la variable CV del nodo considerado igual o sea menor que alfa, puede abandonarse su evaluación.

2. Límite β: el valor de corte para un nodo MAX es un límite superior, denominado beta, igual al valor mínimo de todos los antecesores MIN de dicho nodo. En el momento en que el valor de la variable CV del nodo considerado iguale o sea mayor que beta, puede abandonarse su evaluación.

Las operaciones de poda y actualización de los límites pueden representarse mediante un procedimiento recursivo al que se puede denominar $V(J;\alpha,\beta)$. Este procedimiento toma dos parámetros α y β, que deben cumplir la condición $\alpha<\beta$ y devuelve V(J), el valor minimax del nodo J si éste

se encuentra entre los valores límite α y β. Si no se cumple esta última condición, devuelve α si $V(J)\leqq\alpha$ o β si $V(J)\geqq\beta$.

El algoritmo puede expresarse de la siguiente forma:

1. Si J es un nodo terminal, devolver V(J) = e(J). De lo contrario, sean $J_1, J_2, \ldots, J_b$ los sucesores de J. Asignar $k\leftarrow 1$, si J es un nodo MAX, ir al paso 2; si no, ir al paso 2'.
2. Asignar $\alpha\leftarrow \max[\alpha, V(J_k; \alpha, \beta)]$.
2'. Asignar $\beta\leftarrow \min[\beta; V(J_k; \alpha, \beta)]$
3. Si $\alpha\geqq\beta$, devolver β; de lo contrario, continuar.
3'. Si $\beta\leqq\alpha$, devolver α; de lo contrario, continuar.
4. Si k = b, devolver α; de lo contrario, proceder con J_{k+1}, es decir, asignar $k\leftarrow k + 1$ y pasar a 2.
4'. Si k = b, devolver β; de lo contrario, proceder con J_{j+1}, es decir, asignar $k\leftarrow k + 1$ y pasar a 2'.

La eficiencia de este método de búsqueda depende de la ordenación de los valores terminales. En juegos complejos, la diferencia entre el caso mejor y el peor puede llegar a ser importante, duplicando la profundidad del árbol descendente que ha de explorarse.

El análisis del comportamiento del algoritmo alfa-beta demuestra que permite extender la profundidad de la búsqueda un 33 por 100 más que la que se utiliza con el procedimiento MINIMAX exhaustivo.

Existen otros algoritmos que también son específicos de la búsqueda en árboles de juegos. De ellos los más importantes e interesantes son: la estrategia SSS* y la estrategia SCOUT. Ambos son algoritmos puros de búsqueda que examinan el árbol y podan los nodos menos prometedores utilizando para ello diferentes criterios.

De todos ellos se han realizado estudios comparativos cuyas conclusiones aconsejan la utilización de uno u otro para un determinado tipo de juegos, desaconsejándola en otros casos.



EL METODO GPS

Aunque no es lo más usual, algunos de los problemas que se muestran en este libro pueden ser resueltos mediante otro tipo de técnicas diferentes de las técnicas de búsqueda anteriormente descritas. Uno de los mecanismos que se pueden utilizar es el método GPS. Este método, cuyo nombre proviene de General Problem Solver (Resolutor General de Problemas), fue desarrollado en los años sesenta por Simon, Newel y Shaw. Su propósito, como su propio nombre indica, era encontrar un procedimiento general que pudiera resolver cualquier tipo de problema. El método tuvo mucho éxito en la época de su creación, pero más tarde quedó prác-

ticamente como objeto de estudio, aunque importantes partes de él se han utilizado como base de diversas metodologías posteriores.

El método GPS plantea la búsqueda de solución haciendo uso de lo que se ha venido denominando «Análisis de Medios-Fines». Este análisis propone la descomposición del problema original en varios problemas más simples y sencillos de resolver. Esta descomposición se modeliza mediante un grafo Y/O. Así, pues, el método consiste en tomar la solución como un fin y buscar los medios que permitan llegar a él. La tarea que debe ser realizada por el método debe estar representada en términos de objetos, operadores y diferencias. En cada paso se selecciona un operador en función de las diferencias existentes entre el objeto actual y el deseado. Los operadores pueden estar organizados de diferentes formas, aunque lo más usual es que se encuentren jerarquizados en base al tipo de diferencias que reducen. La elección del operador que ha de utilizarse en cada momento es uno de los problemas fundamentales en la realización del sistema.

LOS SISTEMAS DE PRODUCCION

1. Introducción

Tan pronto como se dieron los primeros pasos en el campo de la IA, se empezó a cuestionar la representación y tratamiento que los algoritmos tradicionales daban a los problemas y a sus soluciones. Casi todos los programas secuenciales son, de alguna forma, dependientes de los datos. Especialmente el flujo de control y la utilización de los datos son fijos, debido a la forma de construir los programas. Los programas tradicionales poseen como característica esencial su secuencialidad. Esto es, ejecutan paso por paso las instrucciones del programa y siempre en el mismo orden. Esto no es un inconveniente para determinadas aplicaciones. Sin embargo, no es adecuado en situaciones en las que el entorno es cambiante y es necesario simular las respuestas humanas a estímulos provenientes del mismo.

En estos casos es necesario que el programa reaccione a cambios del entorno. Si no recibe ningún estímulo, debe avanzar un paso secuencialmente dependiendo de la historia del proceso. Se entiende por historia del proceso al conjunto de acciones que se ejecutaron anteriormente. En caso de que el entorno cambie constantemente, podrá ser útil dividir el programa en una serie de módulos, cada uno de los cuales ha de encargarse de tratar convenientemente estos cambios.

Esto dio lugar a la idea de utilizar los datos como las partes del programa que dirigen las operaciones. En esto radica la diferencia fundamental

con los programas clásicos. En éstos, es el programa el que maneja los datos.

Todas estas ideas fructificaron en la concepción de los Sistemas de Producción que, en general, pertenecen a una clase de sistemas denominados «Sistemas de Inferencia Dirigidos por Patrones».

2. Componentes de un sistema de producción

En condiciones normales, un Sistema de Producción tiene tres componentes básicos:

a) *Un conjunto de datos o Base de Hechos,* que contiene la información del mundo, del entorno y del programa. Estos datos se almacenarán en dicha Base de diferentes formas, dependiendo de la estructura o representación del conocimiento que se utilice. Como ejemplos se pueden citar:

— En el problema del «8-Puzzle» o del «15-Puzzle» la Base de Hechos estará formada por una matriz que representará en cada momento la situación del problema. También se puede almacenar en ella la información referente al estado final o aspectos referentes al aprendizaje.

— En el problema de los misioneros y los caníbales se puede utilizar una estructura en forma de lista con tres elementos: número de misioneros en una orilla, número de caníbales en la misma orilla, y dónde se encuentra la barca. Esto último se representa con una letra, como izquierda y derecha, o por un número, como 1 izquierda y 2 derecha.

Como puede observarse, la estructura de la Base y la forma en que se almacenan los datos es muy variable y se deja a gusto del programador el tipo de representación que quiera utilizar. Esto, por supuesto, tendrá repercusiones en los otros dos componentes del Sistema de Producción, pues se tendrán que ceñir en cierto modo a la estructura elegida.

b) *Un conjunto de reglas o Base de Reglas.* Contienen la mayoría del conocimiento sobre la solución de problemas. Las reglas, también llamadas producciones, constan de dos partes: un lado izquierdo, denominado antecedente o condición; y un lado derecho, llamado consecuente o de acción.

La forma de representarlos gráficamente es:

antecedente → consecuente,

aunque se utilizan muchas otras formas como:

condición → acción,

lo cual explica más sobre la filosofía de las reglas, utilizando encadenamiento hacia adelante, el cual se explicará posteriormente.

Si con los datos disponibles en la Base de Hechos se cumple la condición, entonces se «ejecuta» la parte derecha de acción. Ello quiere decir que se realizan las acciones especificadas en la parte derecha. Estas acciones pueden tener muy diferentes formas y representan la parte activa de las producciones.

La forma general de ejecución de una regla indica cómo se representarán comúnmente en lenguaje de ordenador. Normalmente, para ello se utiliza la instrucción:

IF condición THEN acción,

presente en todos los lenguajes. Tanto la parte de condición como la parte de acción pueden estar formadas por más de un elemento. Por ello se puede expresar también:

IF condición$_1$ condición$_2$... condición$_n$ THEN acción$_1$ acción$_2$... acción$_m$,

lo cual quiere expresar la idea de que si se cumplen las condiciones 1, 2, ... y n, se está en condiciones de realizar las acciones que van de la 1 a la m. Un ejemplo de una regla de este tipo podría ser:

IF "Problema resuelto" THEN Parar,

que expresa la idea de que si se llega al fin del programa, o bien al estado final, se debe parar la ejecución del mismo.

c) *Una estrategia de control, intérprete de reglas o motor de inferencias.* Hasta ahora se ha definido la Base de Conocimiento. Esta está formada por la Base de Hechos y la Base de Reglas. Pero es necesario un sistema capaz de hacerlos funcionar. Esta es la principal misión de la Estrategia de Control. Es la encargada de encadenar los ciclos de trabajo en los que se divide el funcionamiento de un Sistema de Producción. Asimismo, se ocupa de dirigir al sistema dentro de la ejecución de cada ciclo. En su caso más general, posee dos fases en cada ciclo: la fase de Decisión o selección de reglas; y la de Acción, activación, deducción, o ejecución de las reglas elegidas para aplicar. El funcionamiento de dicha estrategia será tratado más adelante, dentro de este mismo capítulo.

3. Tipos de sistemas de producción

Basándose en la sintaxis de las reglas y en su estructura de control, se pueden dividir los Sistemas de Producción en dos tipos:

a) Los sistemas dirigidos por los antecedentes, que son los más utilizados. En éstos, los antecedentes son condiciones que se han de cumplir para

que se puedan ejecutar los consecuentes o acciones. De ello se deriva que sea el antecedente el que gobierne el funcionamiento de la regla. Las condiciones pueden ser meras verificaciones de si determinados datos figuran en la Base de Hechos o no. Pero también pueden ser condiciones más complejas, que envuelvan operaciones con los elementos de la Base.

La estrategia utilizada se denomina encadenamiento hacia adelante, puesto que el sentido de la ejecución es el marcado por la flecha de la regla, es decir, condición→acción. Esto equivale al MODUS PONENS de la lógica.

b) Los sistemas dirigidos por el consecuente. En este tipo tanto el antecedente como el consecuente son aserciones acerca de los datos. La ejecución de las reglas se guía por los consecuentes y retrocede para intentar probar los antecedentes. Por ello, la estrategia utilizada se denomina encadenamiento hacia atrás. Esta estrategia busca las metas en los consecuentes de las reglas y selecciona aquellas reglas que las incluyan. A continuación intenta demostrar los antecedentes de las reglas encontradas. Si lo consigue, el consecuente se puede alcanzar y con él las metas. Debe observarse que el sentido de ejecución va al contrario de los anteriores sistemas. En este caso va de derecha a izquierda, o hacia atrás. Este método se corresponde con el MODUS TOLLENS de la lógica.

En este capítulo se explicará con más detalle el funcionamiento de estas dos estrategias, incluyendo un ejemplo de la primera.

4. Funcionamiento de un sistema de producción en encadenamiento hacia adelante

Como ya se ha explicado, la estrategia de control posee el conocimiento de control del sistema. Este control se lleva a cabo sobre dos áreas del funcionamiento: el encadenamiento de los ciclos de trabajo en los que se divide el proceso de ejecución de un Sistema de Producción, y las tareas que han de realizarse dentro de cada ciclo de trabajo. Cuando se inicia la ejecución del Sistema de Producción, tanto la Base de Hechos como la de Reglas deben contener la información y situación iniciales, así como la descripción de los posibles movimientos o transiciones entre estados. La Estrategia de Control, entonces, debe seleccionar el primer ciclo de trabajo y comenzar su operación. Esta se divide en dos fases: Decisión y Acción.

a) Fase de Decisión o selección de las reglas. Esta fase consta a su vez de las siguientes etapas:

1) Etapa de Restricción.

Esta etapa intenta reducir en lo posible el número de reglas que han de ser examinadas. Para ello, en la medida de lo posible, se debe dividir

la Base de Reglas al principio del proceso en familias y subfamilias de reglas. Esto se debe realizar de modo que dentro de una familia queden las reglas que se apliquen sobre un dominio similar. Este dominio será diferente, de alguna forma, del dominio común referente a otra familia de reglas. Una familia se puede subdividir también en subfamilias, y así sucesivamente.

Por ejemplo, si el sistema trata de enfermedades, será conveniente dividir el dominio de las enfermedades en subdominios. Así, se podrán dividir las reglas en dominios de reglas que traten las enfermedades del estómago, las del corazón, etc. Si el dominio es el de los juegos, también se pueden dividir las reglas en subdominios. Las reglas concernientes a la salida de determinados juegos como el ajedrez y el dominó se pueden separar de las concernientes al desarrollo normal del juego.

Debido a estas divisiones puede efectuarse esta etapa de restricción. En ella se selecciona el dominio sobre el que se va a trabajar en cada ciclo. Por tanto, la Estrategia de Control trabaja en las etapas siguientes con un subconjunto de la Base de Reglas, lo cual disminuye considerablemente el tiempo de ejecución.

Después de esta etapa y para el funcionamiento del ciclo en que se encuentre la ejecución del sistema, se dispondrá de una Base BR_1 igual o menor a la anterior:

$$BR_1 \subseteq BR.$$

Independientemente de esta división de la Base de Reglas, se puede realizar también una división de la Base de Hechos. Esto facilitaría la tarea en la etapa de equiparación. Al dividirla se dispondría de una Base de Hechos BH_1 igual o menor a la anterior:

$$BH_1 \subseteq BH.$$

2) Etapa de Filtrado, Equiparación o Cotejo («Pattern Matching»).

En esta etapa se recogen las reglas cuya parte de condición se satisface con los datos que, en ese momento, estuvieran en la Base de Hechos. Para ello se realiza la equiparación de las condiciones de las reglas con la información contenida en la Base de Hechos. Esto causa que de la BR_1 (restricción de la Base de Reglas original) se pase a una nueva BR_2, también llamada conjunto conflicto, tal que:

$$BR_2 \subseteq BR_1$$

Por tanto, cuanto menor sea el número de reglas que haya en BR_1, más rápida se efectuará esta etapa, ya que habrá que comprobar menos reglas. A su vez, cuanto menor sea el número de las reglas del conjunto conflicto, menos trabajo tendrá que realizarse en la siguiente etapa. Por todo esto se

explica que sea necesario tener una buena heurística o información *a priori*, de forma que en cada etapa se reduzca considerablemente el número de reglas y hechos sobre los que se ha de trabajar. Comúnmente, esta etapa es la que más recursos computacionales consume.

3) Etapa de Resolución de Conflictos.

Con las dos etapas anteriores se ha logrado obtener el conjunto de las reglas que, en cada ciclo, son susceptibles de aplicación. Ello comporta la idea de que se verifican todas las condiciones de cada regla y, por tanto, el sistema está en condiciones, está en disposición, de ejecutar las partes derechas de las reglas.

Pero aún no se ha llegado al final del proceso de Decisión. Se debe restringir el número de reglas a una o a un número muy bajo. Esta decisión la lleva a cabo la Estrategia de Control. Y en esta etapa, fundamentalmente, es en lo que difieren las diversas estrategias. Existen muchas opciones, métodos y técnicas. Algunos de los más utilizados son:

— Ejecutar la primera regla del conjunto conflicto.

— Ejecutar la regla que incorpore más conocimiento a la Base de Hechos.

— Ejecutar la regla con más prioridad. Esta prioridad la establecerá el diseñador del Sistema de Producción. Vendrá dada en función de la situación global del sistema y de la importancia relativa de aplicar una regla antes que otra.

— Ejecutar la regla más específica. Por ejemplo, que incorpore las restricciones más fuertes.

— Ejecutar la regla que concierne al elemento añadido más recientemente al contexto o a la Base de Hechos.

— Ejecutar una nueva regla, es decir, que no se haya aplicado antes o, por lo menos, recientemente. También se podrá considerar el caso contrario: la regla que más veces se haya ejecutado.

— Utilizar una función heurística que determine qué regla ha de aplicarse a continuación. Se pueden utilizar para ello algoritmos que obtengan caminos óptimos entre la situación del momento y la situación final.

— Ejecutar una regla arbitraria. Esto puede parecer poco efectivo. Sin embargo, al terminar la segunda etapa se supone que las reglas pertenecientes al conjunto conflicto son teóricamente iguales desde el punto de vista de la posibilidad de aplicación.

— Por último, se pueden explorar exhaustivamente todas las alternativas posibles y ejecutar todas las reglas. Esto recibiría un gran apoyo si se dispusiera de máquinas trabajando en paralelo.

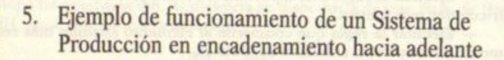
Con esta etapa se ha concluido el análisis de la fase de Decisión. En este momento se dispone de una o varias reglas que se han de aplicar.

b) Fase de Acción.

Esta fase sólo comprende una etapa y consiste en ejecutar las acciones especificadas en las partes derechas de las reglas elegidas. Unas veces estas acciones se limitarán a añadir, eliminar o modificar los elementos de la Base de Hechos. Otras veces serán programas escritos en LISP o cualquier otro lenguaje, que permitirán realizar funciones de entrada/salida, operaciones numéricas, etc.

La ejecución del Sistema de Producción termina cuando se introduce en la Base de Hechos el hecho fijado como meta, o bien cuando así lo exprese la acción de una de las reglas. En ocasiones no se alcanza la solución debido a que se ha aplicado una regla que no lleva a ningún sitio. En estos casos se utiliza la técnica de «backtracking», retroceso o de marcha atrás. Su aplicación significa que el sistema vuelve sobre sus pasos hasta el momento en el que se eligió una regla del conjunto conflicto antes que otra u otras. El estado del sistema ha de ser el mismo que cuando se efectuó dicha elección. A continuación, se elige otra regla y se sigue el proceso.

El encadenamiento hacia adelante presenta problemas como la no focalización hacia la meta. Para mejorar este aspecto, ha de construirse una Estrategia de Control que dirija al sistema hacia la meta establecida. Será importante, por tanto, que en la fase de Decisión se decida aplicar la regla que más rápido lleve a la solución. Otro de los inconvenientes que presenta este tipo de estrategia es que, al principio, han de estar almacenados los datos iniciales en la Base de Hechos.



5. Ejemplo de funcionamiento de un Sistema de Producción en encadenamiento hacia adelante

El Sistema de Producción sobre el que basaremos el ejemplo es el siguiente:

— Base de Reglas:

Estará formada por las siguientes reglas:

1. SI P' ENTONCES B, D', E'
2. SI A', B ENTONCES D', G'
3. SI A' ENTONCES C, P'
4. SI D', L ENTONCES C', I
5. SI E' ENTONCES H, D'
6. SI H' ENTONCES L, A'

La interpretación de estas reglas puede ser desde la identificación de una enfermedad hasta la clasificación de un elemento químico.

— Base de Hechos:

Estará formada por los siguientes hechos:

BH = <H', B, C, J>

Tanto en la Base de Reglas como en la Base de Hechos se toma la convención de que:

— las letras con apóstrofe (H') son hipótesis cuya certeza ha de demostrarse.

— las letras sin apóstrofe (H), sin embargo, son hechos ciertos ya comprobados.

— Estrategia de Control:

a) Fase de Decisión: En este problema no se va a efectuar etapa de restricción. La etapa de equiparación va a consistir en la confrontación de los elementos de los antecedentes de las reglas con los elementos de la Base de Hechos. De ella resultará el conjunto conflicto, formado por las reglas cuyos antecedentes pertenecen por completo a la Base de Hechos. En cuanto a la etapa de resolución del conjunto conflicto se explicarán dos de los métodos más comunes.

b) Fase de Acción: Existen cuatro posibles acciones a ejecutar con los datos del consecuente:

— Si en el consecuente de la regla aparece un hecho que no perteneciera ya a la Base de Hechos, se le incluye directamente en la misma.

— Si un hecho aparece como cierto (sin apóstrofe) en el consecuente y en la Base aparece como hipótesis (con apóstrofe) se retirará de la Base como hipótesis y se incluirá como cierto.

— Si un hecho aparece como hipótesis en el consecuente de la regla y como cierto en la Base de Hechos, no se realizará ninguna acción.

— Si un hecho aparece de la misma forma en la Base y en el consecuente de la regla, tampoco se realizará ninguna acción.

Por otra parte, con los elementos del antecedente también se realiza una determinada acción. Esta consiste en que si el elemento aparece como hipótesis en el antecedente, se eliminará de la Base de Hechos.

Se supondrá que el proceso termina cuando se introduzca el hecho H en la Base y todos los demás datos de la Base aparezcan como ciertos en ella.

A continuación se detalla el funcionamiento para dos técnicas diferentes de resolución del conjunto conflicto.

a) *Elección de la primera regla del conjunto conflicto.*

Primer ciclo:

Como se ha dicho, no se va a considerar la etapa de restricción en este ejemplo, por lo que pasa directamente a la etapa de equiparación. Los pasos a ejecutar serán:

— Se escoge la primera regla SI P' ENTONCES B, D', E' y se observa si los datos del antecedente pertenecen a la Base de Hechos. En este caso sólo hay un dato, P', y no pertenece a la Base. Debido a esto no se introduce la regla 1 en el conjunto conflicto.

— Se realiza el mismo proceso con la regla 2. Se comprueba que B pertenece a la Base. Pero A' no pertenece a la Base. Por tanto, tampoco se introduce la regla 2 en el conjunto conflicto.

— Se continúa este proceso con todas las reglas. Al final, se tendrá que la única regla del conjunto conflicto es la regla 6, ya que su antecedente, H', pertenece a la Base de Hechos.

La etapa de resolución del conjunto conflicto es nula, ya que sólo existe una regla en el conjunto conflicto, que es la que se selecciona para ser aplicada en la segunda fase.

En la fase de Acción se realizarán las siguientes tareas:

— Se escoge el primer elemento del consecuente de la regla L. Como L no pertenece a la Base de Hechos, se introduce directamente.

— A continuación se elige el segundo elemento del consecuente, A', que, en este caso, es el último. Al no pertenecer tampoco a la Base de Hechos, se introduce también en la Base.

— Por último, debido a que H' era una hipótesis, se la retira de la Base de Hechos.

La Base de Hechos, después del primer ciclo, ha quedado como sigue:

BH = <A, L, B, C, J>

El orden en el que aparecen los hechos en la Base de Hechos es irrelevante en este ejemplo. Las acciones realizadas en este primer ciclo se pueden expresar en una tabla, como la 1:

Base de Hechos	Regla aplicada
H' B C J	6
A' L B C J	

Tabla 1

Con esto se cierra el primer ciclo de funcionamiento:

Segundo ciclo:

Dentro de la etapa de equiparación, se observa que la primera regla no se puede introducir en el conjunto conflicto, puesto que P' no pertenece a la Base de Hechos. Sin embargo, la segunda regla sí se puede introducir, puesto que A' pertenece a la Base y B también pertenece.

La tercera regla también se puede introducir en el conjunto conflicto, debido a que A' pertenece a la Base. A partir de la tercera regla ninguna más se introducirá en el conjunto. Por tanto, el conjunto conflicto estará formado por las reglas 2 y 3.

En la etapa de resolución del conjunto conflicto ha de decidirse qué regla será la que se active. En este caso, la estrategia escoge la primera regla del conjunto conflicto, que es la 2.

En la fase de acción de este ciclo se introducen D' y G' en la Base de Hechos, ya que no pertenecían a la Base. Al mismo tiempo se extrae A' de la Base, pues aparece en el antecedente de la regla 2 como hipótesis. Después de este ciclo la tabla quedará como aparece en la tabla 2.

Base de Hechos	Conjunto conflicto	Regla aplicada
H' B C J	6	6
A' L B C J	2 3	2
D' G' L B C J		

Tabla 2

Si se realiza otro ciclo sobre el Sistema de Producción, se obtendrá la tabla 3.

Base de Hechos	Conjunto conflicto	Regla aplicada
H' B C J	6	6
A' L B C J	2 3	2
D' G' L B C J	4	4
I G' L B C J		

Tabla 3

En este punto se observa que no se puede seguir el proceso. No hay ninguna regla cuyo antecedente pertenezca a la Base de Hechos. Con lo cual, en la fase de equiparación, se obtiene un conjunto conflicto nulo. Esto in-

dica que por este camino no se va a llegar a una solución. Como H no pertenece a la Base de Hechos, tampoco es una situación final, por lo que no queda más remedio que echar marcha atrás. Se retrocede hasta la ocasión más cercana en la que se dispusiera de un conjunto conflicto de más de una regla. Este es el caso del segundo ciclo, donde el conjunto conflicto estaba formado por las reglas 2 y 3. El proceso vuelve a ese punto, por lo que la Base de Hechos debe ser la que se tenía antes de ejecutar la regla 2. A continuación se ejecuta la regla 3, en vez de la regla 2, y se continúa el proceso.

Siguiendo el funcionamiento del Sistema de Producción, se obtiene la tabla adjunta:

Base de Hechos	Conjunto conflicto	Regla aplicada
H' B C J	6	6
A' L B C J	2 3	3
P' L B C J	1	1
D' E' L B C J	4 5	4
I E' L B C J	5	5
H D' I L B C J	4	4
H I L B C J		

Tabla 4

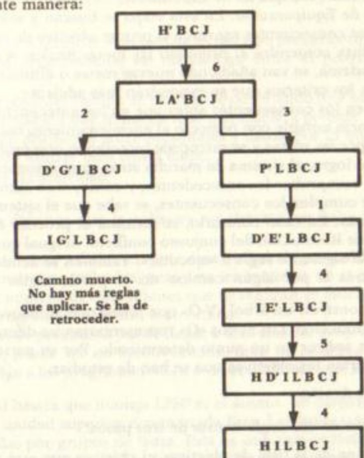
Se ha llegado a la solución, puesto que H pertenece a la Base de Hechos y todos los hechos de ésta son ciertos. Sin embargo, ha sido necesaria la utilización del «backtracking». Supóngase que se utiliza otra Estrategia de Control.

b) *Elección de la regla que más conocimiento cierto incorpore a la Base de Hechos.*

En este caso, la Estrategia de Control hubiera elegido la regla 3 en lugar de la 2 en el segundo ciclo. Esto se debe a que la regla 3 incorpora un conocimiento cierto, C, y la 2 no incorpora ninguno. Al final del proceso mediante esta Estrategia de Control, se obtendría directamente la tabla 4, sin necesidad de realizar la marcha atrás.

Existen otras formas más utilizadas para representar la ejecución de un Sistema de Producción. Una de las más comunes es la que la expresa en

forma de árbol. Así, el funcionamiento por la técnica a) vendría expresado de la siguiente manera:



Los arcos entre los nodos del árbol representan la regla aplicada, y los nodos representan el estado de la Base después de aplicar la regla. Como puede observarse, se ha representado la marcha atrás que el sistema tuvo que efectuar.



6. Funcionamiento de un Sistema de Producción en encadenamiento hacia atrás

Los Sistemas de Producción que funcionan con este tipo de encadenamiento son también llamados dirigidos por las metas. Esto es debido a que en lo primero que se fijan es en las metas u objetivos. En general, el funcionamiento de un Sistema de Producción de este tipo es el siguiente:

1) *Fase de Decisión*

a) Etapa de Restricción. Al igual que en los anteriores, se puede efectuar una etapa de restricción tanto de la Base de Hechos como de las Re-

glas. Esto, como ya se ha explicado, facilita la labor a las siguientes etapas, pues limita el universo que ha de examinarse.

b) Etapa de Equiparación. En esta etapa se buscan y seleccionan las reglas en cuyos consecuentes aparezca el primer objetivo de la lista de objetivos. Esta lista contendrá al principio las metas finales. A medida que funciona el sistema, se van añadiendo nuevas metas o eliminando las que tuviera, según los criterios que se expondrán más adelante.

Al buscar en los consecuentes antes que en los antecedentes, introduce una diferencia notable con respecto al encadenamiento hacia adelante. Se comienza por las metas y se retrocede intentando comprobar los antecedentes. Si se logra, el sistema da marcha atrás en el razonamiento: si ha sido capaz de comprobar los antecedentes, y como, si se cumplen los antecedentes, se cumplen los consecuentes, se sabe que el sistema puede alcanzar las metas. En caso contrario, se termina el proceso con fracaso.

c) Etapa de Resolución del conjunto conflicto. Al igual que antes, esta etapa escoge la siguiente regla a «ejecutar». También se acude al método de marcha atrás si por algún camino no se puede seguir el proceso.

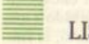
Se podría construir un árbol «Y-O» que reflejara el funcionamiento del Sistema de Producción. Los nodos «O» representarían las diferentes reglas que se pueden aplicar en un punto determinado. Por su parte, los nodos «Y» representarían los objetivos que se han de estudiar.

2) *Fase de Acción*

La ejecución de una regla consta de tres pasos:

— Se elimina de la lista de objetivos el objetivo que esté a la cabeza.
— Se comprueba si los hechos del antecedente de la regla pertenecen a la Base de Hechos.
— Se introducen en la lista de objetivos los hechos del antecedente que no estuvieran en la Base de Hechos. Normalmente, se introducen al principio de dicha lista, pero en ocasiones puede interesar otro tipo de ordenación de la lista.

A continuación se empezaría otro ciclo con la nueva lista. Cuando la lista de objetivos se quede vacía, termina el proceso con éxito, pues han podido deducirse los objetivos.



LISP

En este apartado se van a describir las principales características de LISP. Es un lenguaje que nació en los años 60. LISP toma su nombre de «List programming» (programación con listas) y está fundamentalmente orientado a la programación simbólica. Esta es una de las características

que han hecho que LISP sea el principal lenguaje de programación que se utiliza en la IA. Otras características que le hacen apropiado para IA son:

— LISP es un lenguaje interactivo: es del tipo pregunta-respuesta, convirtiéndose la programación en un diálogo entre el programador y la máquina.
— Con el paso del tiempo se han ido desarrollando una serie de entornos de programación que lo hacen más manejable y potente.
— No hace distinción entre programas y datos, por lo que un programa LISP puede utilizar otro como datos.
— Admite recursividad.

Entre los defectos que le achacan sus detractores cabe destacar dos:

— LISP no tiene suficientemente desarrollado la parte dedicada a operaciones aritméticas y cálculos, pero esto, en realidad, no es un defecto, ya que en IA el número de operaciones que se realizan es muchísimo menor que en un programa de gestión o científico.
— El uso de paréntesis puede llegar a confundir al programador. Esto, en principio, puede ser un inconveniente, pero no es difícil de dominar. Además, obliga a los programadores a concebir claramente sus programas.

La unidad básica que maneja LISP es el átomo. Un conjunto de átomos forman una unidad superior denominada lista. Las listas también pueden estar formadas por grupos de listas. Esta es una característica fundamental de LISP. Los átomos y las listas se llaman expresiones simbólicas.

Ejemplos de átomos son:

27
3.1415
TRES
B27

Los dos primeros se denominan átomos numéricos. Los tres restantes son símbolos.

Una lista está compuesta por un paréntesis de apertura, a continuación una serie de uno o más átomos o listas y, por último, un paréntesis de cierre. La lista vacía también es considerada como lista. Por ejemplo:

(UNO DOS TRES)
(UNO (DOS TRES) CUATRO)
( )

La segunda lista consta de tres elementos: UNO (DOS TRES) y CUATRO; y no de cuatro elementos, como se podría pensar en un principio. LISP considera como elementos de la lista los átomos y las listas que lo componen, sin entrar en la estructura de dichas listas.

Otro concepto importante de LISP es el de procedimiento o función: es la entidad básica para decir al intérprete lo que se quiere realizar. Por ejemplo: una función como '* 'multiplica las expresiones que encuentre a continuación. Las funciones que se encuentran predefinidas en el sistema, y que forman la base de LISP, se denominan primitivas.

Por último, un conjunto de procedimientos o funciones forman un programa, que no es otra cosa que una función que utiliza otras funciones.

Primitivas aritméticas

LISP tiene implementadas las cuatro reglas básicas:

— Suma:

(+ 2 3)
  5

— Resta:

(- 7 4)
  3

— Multiplicación:

(* 5 3)
  15

— División

(/ 7 2)
  3.5

Además de estas primitivas, también se pueden utilizar otras como:

— MAX: Halla el máximo de una serie de números.
— MIN: Halla el mínimo de una serie de números.

(MAX 2 4 5 9 1)
  9
(MIN 2 4 5 9 1)
  1

— EXP: eleva una base a un exponente.Tiene la siguiente sintaxis:
(EXP<base><exponente>)
— SQRT: Halla la raíz cuadrada de un número:
(SQRT 16)
  4.0

Si tenemos una expresión más compleja como:

(+(/6 3) (– 5 2)

Se puede deducir fácilmente el resultado: primero se divide 6/3, obteniendo 2.0, y este resultado se suma al obtenido al restar 5 – 2, en total, 5.0. Un proceso semejante a éste es el que sigue el intérprete de LISP al evaluar una expresión como ésta. El proceso será explicado posteriormente.

Primitivas para el manejo de listas

Las dos principales primitivas para trabajar con listas son CAR y CDR.

— CAR: aplicado a una lista devuelve el primer elemento de dicha lista como resultado.

— CDR: aplicado a una lista devuelve otra lista que contiene la original menos el primer elemento:

(CAR '(A B C))
A
(CAR '((A B) C))
(A B)
(CDR '(A B C))
(B C)
(CDR '(A (B C)))
((B C))

La comilla que antecede a las listas tratadas indica al intérprete de LISP que la expresión que viene a continuación no debe ser evaluada.

LISP también permite asignar valores a variables, para ello se utiliza la función SETQ, que tiene la siguiente sintaxis:

(SETQ <variable><valor asignado>)
SETQ devuelve el valor asignado a la variable.
(SETQ ANIMALES '(PERRO GATO CABALLO))
(PERRO GATO CABALLO)

Si ahora se escribe en la pantalla

ANIMALES

El intérprete devolverá el valor asignado a la variable

(PERRO GATO CABALLO)

Esto no descarta la posibilidad de asignar el átomo ANIMALES a otra variable:

(SETQ SER-VIVO 'ANIMALES)
ANIMALES
SER-VIVO
ANIMALES

Si se aplica la función CAR

(CAR ANIMALES)
GATO
(CAR 'ANIMALES)
ERROR

En el primer caso, ANIMALES no lleva comilla y, por tanto, el intérprete lo toma como una variable y busca su valor, aplicando la función CAR a dicho valor. En el segundo caso, al tener comilla toma ANIMALES como una constante y aplicada la función CAR al átomo ANIMALES, produciendo un mensaje de error, ya que no es una lista.

Primitivas para la construcción de listas

Mientras que CAR y CDR descomponen una lista en sus componentes básicos, las siguientes primitivas construyen listas a partir de expresiones simbólicas.

— APPEND: une los elementos de las listas que se pasan como argumentos:

(APPEND '(A B C) '(D E) '(F))
(A B C D E F)

Los argumentos deben ser listas, ya que si no lo son se producirá un error.

(APPEND '((A B) C) '(D (E)))
((A B) C D (E))

– LIST: es parecida a APPEND, ya que también forma listas, pero une las listas que se pasan como argumentos y no los elementos que componen estas listas.

(LIST '(A B) '(C D))
((A B) (C D))
(LIST '(A B C) '(D E) '(F))
((A B C) (D E) (F))

Es decir, cada lista que se pasa como argumento es un elemento de la lista resultante, mientras que en el caso de APPEND cada elemento de las listas que se pasan como argumentos es un elemento de la lista resultante.

— CONS: Toma una lista y añade un elemento al principio de la lista:
(CONS ⟨ nuevo primer elemento ⟩ ⟨ lista ⟩)
(CONS 'A '(A B C))
(A B C)
(CONS '(A) '(B C))
((A) B C)

A continuación se comparan las tres funciones aplicadas a los mismos argumentos:

(APPEND '(A) '(B C D))
(A B C D)
(LIST '(A) '(B C D))
((A) (B C D))
(CONS '(A) '(B C D))
((A) B C D)

También existen en LISP otras funciones auxiliares para manejar listas que se enumeran a continuación:

— LENGTH: devuelve el número de elementos que contiene una lista:
(LENGTH '(A B C D))
4
(LENGTH '((A B) (C D)))
2
(LENGTH '((A B C D)))
1

— REVERSE: coloca en orden inverso los elementos de una lista:
(REVERSE '((A B) C (D (E))))
(((E) D) C (A B))

— SUBST: sustituye, en una lista, un elemento por otro. Sólo la primera vez que lo encuentra:

(SUBST ⟨ nueva expresión ⟩ ⟨ expresión a reemplazar ⟩ ⟨ lista ⟩)
(SUBST 'A 'B '(A B C))
(A A C)

Si no encuentra el átomo indicado, no realiza ninguna operación:
(SUBST 'A 'B '(A (B) C))
(A B C)

— LAST: devuelve una lista formada por el último componente de la lista pasada como argumento:
(LAST '(A B C D))
(D)

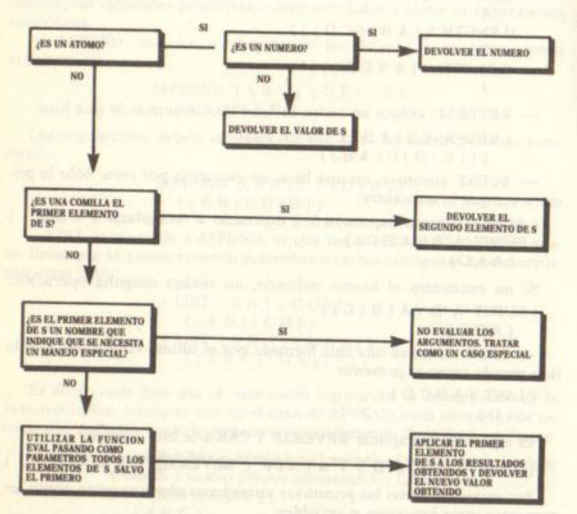
Es equivalente a aplicar REVERSE Y CAR a la lista original:
(LAST '(A B C D)) ≡ (CDR (REVERSE '(A B C D)))

Por supuesto, todas las primitivas vistas hasta ahora aceptan como argumentos otras funciones o variables:

(CAR (CDR '(A (B C) D)))
(B C)

(SETQ DIGITOS '(UNO DOS TRES CUATRO CINCO SEIS SIETE OCHO NUEVE CERO))
(UNO DOS TRES CUATRO CINCO SEIS SIETE OCHO NUEVE CERO)
(SETQ VOCALES '(A E I O U))
(A E I O U)
(APPEND (CDR DIGITOS) VOCALES)
(DOS TRES CUATRO CINCO SEIS SIETE OCHO NUEVE DIEZ A E I O U)
(APPEND (LIST (CAR DIGITOS)) (LAST DIGITOS))
(UNO CERO)

Para conseguir evaluar estas expresiones, el intérprete de LISP sigue los pasos detallados en la figura:



LISP también permite la definición de nuevas funciones que no se diferenciarán en nada de las primitivas, a la hora de operar con ellas. Para

ello se dispone de la primitiva DEFUN. Consiste en combinar, según se necesite, las funciones ya existentes. El formato de DEFUN es el siguiente:

(DEFUN ⟨ nombre-función ⟩ (parámetro$_1$ parámetro$_2$ ... parámetro$_n$)
⟨ acción$_1$ ⟩
⟨acción$_2$⟩
. . .
⟨acción$_n$⟩)

DEFUN no evalúa sus argumentos, sólo sirve para que el intérprete almacene la definición de la función con vistas a su posterior utilización. DEFUN devuelve el nombre de la función como resultado. A partir de ese momento se puede utilizar la nueva función como si de una primitiva se tratase.

Por ejemplo, una función que convierta kilómetros por hora a metros por segundo:

```
( DEFUN CONVERSION ( VALOR )
   ( / ( * VALOR 10 ) 36 )  )
```

Ahora se puede utilizar como una primitiva normal

(CONVERSION 50)
13.8889

Las variables que se declaran como argumentos en la función sólo existen mientras se esté ejecutando la función. Es decir, al evaluar LISP la expresión (CONVERSION 50) asigna 50 a la variable VALOR y realiza las operaciones indicadas en el cuerpo de la función, pero una vez terminadas estas operaciones, la variable desaparece. Si intentamos buscar su valor:

VALOR
ERROR

el intérprete responderá con un error, ya que no es una variable que esté definida fuera de la función. El siguiente ejemplo muestra una función con dos parámetros:

```
( DEFUN FUERZA ( MASA ACELERACION )
( * MASA ACELERACION )  )
```

(FUERZA 5 6)
30

Predicados

Un predicado en LISP es una función que devuelve verdadero o falso como resultado. El equivalente a verdadero en LISP es T (de TRUE, en español, cierto) y el de falso es NIL (en español nada). A efectos de computación, LISP considera verdadero cualquier cosa distinta de NIL. Los predicados principales son:

- ATOM: comprueba si el argumento es o no un átomo:

(ATOM 'UNO)
T
(ATOM VOCALES)
NIL
(ATOM 'VOCALES)
T

— LISTP: comprueba si el argumento es una lista:

(LISTP VOCALES)
T
(LISTP '(A B C D))
T
(LISTP 'A)
NIL

— EQUAL: comprueba la igualdad de los argumentos.

(EQUAL 'B 'B)
T
(EQUAL VOCALES'VOCALES)
NIL
(EQUAL VOCALES '(A E I O U)·)
T

Cuando se utilizan expresiones aritméticas se usa el predicado '=', que comprueba si dos números son iguales.

— NULL: devuelve T si el argumento es una lista vacía y NIL en caso contrario:

(NULL ())
T
(NULL DIGITOS)
NIL

— MEMBER: se utiliza para saber si un átomo forma parte de una lista. Tiene la siguiente sintaxis:

(MEMBER ⟨átomo⟩ ⟨lista⟩)
(MEMBER 'DOS VOCALES)
NIL

En el caso de que se cumpla el predicado, devuelve el resto de la lista a partir del elemento.

(MEMBER 'DOS DIGITOS)
(DOS TRES CUATRO CINCO SEIS SIETE OCHO NUEVE CERO)

— NUMBERP: compruebe si el argumento es, o no, un número:

(NUMBERP 10)
T
(NUMBERP 'DIEZ)
NIL

Por último, existen otra serie de predicados propios de los números como son: '(' (menor que), ')' (mayor que), que comprueban si la sucesión formada por los argumentos es creciente o decreciente.

((2 4 6 8)
T
((2 2 4 6 8)
NIL

ZEROP, EVENP y MINUSP comprueban si el argumento es cero, par o negativo, respectivamente.

Operadores lógicos

A veces es necesario combinar varios de estos predicados para formar la condición que se quiere utilizar; esto se puede realizar con los operadores lógicos.

— NOT: niega el argumento

(NOT T)
NIL
(NOT (NUMBERP 'DIEZ))
T

— AND: sólo es cierto si todos los argumentos lo son. En cualquier otro caso, el resultado es NIL.

— OR: es cierto en cuanto uno de los argumentos lo sea. En cualquier otro caso, el resultado es NIL.

Estos operadores devuelven el último argumento computado como cierto.

(AND (MEMBER 'DOS DIGITOS) (MEMBER 'CERO DIGITOS))
(CERO)
(OR (MEMBER 'DOS DIGITOS) (MEMBER 'SIETE DIGITOS))
(SIETE OCHO NUEVE CERO)
(OR (NOT (NUMBERP 'DIEZ)) (LISTP VOCALES) (MEMBER

'ROJO DIGITOS))
T

— COND: es un predicado que se utiliza para seleccionar una alternativa de entre varias posibles; su forma general es:

```
( COND    ( ⟨TEST1⟩      ⟨ACCION1⟩ )
          ( ⟨TEST2⟩      ⟨ACCION2⟩ )
          ...............  ...............
          ( ⟨TESTN⟩      ⟨ACCIONN⟩) )
```

Cada lista es una opción, el mecanismo que se sigue para ejecutar este predicado es el siguiente: se va comprobando cada test hasta que se cumple uno, entonces se ejecuta la acción correspondiente a dicho test y se termina la función. COND devuelve como resultado el que devuelva la última función evaluada dentro del predicado; en caso de que ningún test se cumpla, el predicado devuelve NIL y no ejecuta ninguna acción.

```
( DEFUN NUM-DIAS ( MES )
( COND    ( ( MEMBER MES '( ENERO MARZO MAYO JULIO AGOSTO
               OCTUBRE DICIEMBRE ) )
          31 )
          ( ( EQUAL MES 'FEBRERO )
          28 )
          ( ( MEMBER MES '( ABRIL JUNIO SEPTIEMBRE NOVIEMBRE ))
          30 )  )  )
```

(NUM-DIAS'MARZO)
31
(NUM-DIAS'CUATRO)
NIL

ya que no se cumple ninguna condición y, en consecuencia, no se realiza ninguna acción.

Si se quiere que dé un mensaje de error en caso de que no se cumpla ninguna condición, o que se realice alguna acción por defecto, se puede utilizar el valor T:

```
( DEFUN NUM-DIAS ( MES )
( COND    ( ( MEMBER MES '( ENERO MARZO MAYO JULIO AGOSTO
               OCTUBRE DICIEMBRE ) )
          31 )
          ( ( EQUAL MES 'FEBRERO )
          28 )
          ( ( MEMBER MES '( ABRIL JUNIO SEPTIEMBRE NOVIEMBRE ))
          30 )
          ( T '( EL VALOR ASIGNADO NO ES UN MES ) )  )  )
```


Ahora la respuesta a (NUM-DIAS'CUATRO) será:

EL VALOR ASIGNADO NO ES UN MES

Todas las funciones que se han explicado hasta el momento, salvo las que se dice explícitamente, pueden aceptar un número variable de argumentos. Los operadores AND y OR también aceptan un número variable de predicados.

Las funciones CAR y CDR se pueden combinar de la siguiente manera:

CADR...=(CAR (CDR ...)) CDDAR ... = (CDR (CDR (CAR ...)))

hasta 4 niveles de paréntesis.

Variables libres y ligadas

Cuando se utiliza una función, se hace una serie de asignaciones a los argumentos que tiene dicha función. Esta asignación recibe el nombre de «asignación lambda». Por ejemplo:

(SETQ VAL 125)
125
(SETQ AUX 18)
18
(SETQ CIEN 100)
100

```
( DEFUN DUPLICAR ( VAL )
( SETQ VAL ( * 2 VAL ) )
( SETQ AUX VALOR )  )
```

si se llama a la función duplicar con el argumento cien

(DUPLICAR CIEN)
200

ahora las variables tienen los siguientes valores:

AUX
200
CIEN
100
VALOR
125

El valor de CIEN nunca cambia, ya que no se hace ninguna nueva asignación a CIEN. El valor de AUX siempre va a cambiar, ya que se hace una

asignación dentro de la función. El valor contenido en VAL sí que va a cambiar durante el proceso, ya que al entrar en la función se le asigna 100, luego en virtud de las operaciones pasa a ser 200 y al salir de la función se restaura el valor primitivo, ya que VAL está definida en la lista de parámetro.

El proceso que se sigue es el siguiente: al hacer la llamada a una función, LISP crea espacio para almacenar los valores asignados a los argumentos de la función. Cada vez que aparezca una referencia a las variables que están contenidas en la lista de argumentos, se utiliza el valor almacenado en dichos espacios de memoria, sin alterar el contenido de alguna variable que tuviese el mismo nombre, pero que estuviese definida a un nivel superior. Cuando se termina de ejecutar la función, la zona de memoria queda libre. De este proceso surge el concepto de variable libre y ligada.

— Variable libre, con respecto a una función, es un símbolo que no aparece en la lista de argumentos de la función.

— Variable ligada, con respecto a una función, es la que aparece en dicha lista.

Recursión

La recursión es una de las ideas que más potencia da a LISP y una de las razones por las que se utiliza LISP en IA (muchos problemas se resuelven con recursión). Se dice que un objeto es recursivo cuando forma parte de sí mismo o se define en función de sí mismo. Un ejemplo es la imagen de televisión que se contiene a sí misma. También es un concepto que aparece con frecuencia en matemáticas, por ejemplo, en la definición de funciones:

El factorial de un número es ese número multiplicado por el factorial de su antecesor.

$$FAC(N) = N * FAC(N-1)$$
$$FAC(0) = 1$$

La potencia de la recursividad reside en poder definir un número infinito de objetos mediante un enunciado finito o, lo que es lo mismo, un número infinito de operaciones en una operación finita. A pesar de su potencia, la recursión es un arma de doble filo y en determinadas ocasiones puede suponer un mayor gasto de tiempo que una solución no recursiva del problema, ya que hay problemas en los que la recursión hace repetir cálculos inútilmente, aumentando el tiempo de computación.

Lo siguiente es un ejemplo de cómo se trata la recursión en LISP.

```
( DEFUN  FACTORIAL ( N )
      ( COND ( ( EQUAL N 0 ) 1 )
            ( T ( * N (FACTORIAL ( - N 1 ) ) ) ) )  )  )
```

(FACTORIAL 6)
24

Al efectuar esta operación, LISP sigue los siguiente pasos.

Otro ejemplo:

```
(DEFUN N-ESIMO ( N LISTA )
      ( COND  ( ( NOT (ATOM LISTA ) )
            ( COND ( ( EQUAL 0 ( - N 1 ) ) ( CDR LISTA ) )
                  ( T ( N-ESIMO ( - N 1 ) ( CDR LISTA ) ) ) )  )  )  )
```

(N-ESIMO 3 '(UNO DOS TRES CUATRO CINCO SEIS))
(TRES CUATRO CINCO SEIS)

Los pasos que sigue son los siguientes:
Se puede demostrar que cualquier operación recursiva puede ser realizada de forma iterativa.

Iteración

LISP posee una serie de primitivas que permite realizar iteraciones:

— MAPCAR: se utiliza cuando una misma función va a ser utilizada sobre toda una lista de elementos. Su sintaxis es la siguiente:

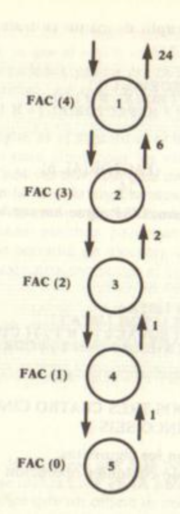
(MAPCAR ⟨función⟩ ⟨lista⟩)

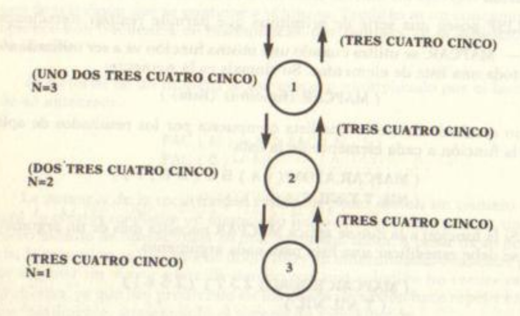
como resultado devuelve una lista compuesta por los resultados de aplicar la función a cada elemento de la lista.

(MAPCAR ATOM'((A) B C (D E) F))
(NIL T T NIL T)

Si la función a la que se aplica MACPAR necesita más de un argumento, se debe especificar una lista para cada argumento.

(MAPCAR EQUAL (2 3 7) '(2 5 6))
(T NIL NIL)





— APPLY: se utiliza para aplicar una función a una serie de argumentos. Considérese el siguiente caso:

```
( SETQ NUM'( 2 4 8 ) )
( 2 4 8 )
```

Si ahora se intenta hacer

```
( + NUM )
```

el intérprete va a responder con un mensaje de error, ya que al evaluar la función '+', espera encontrar argumentos numéricos, y lo que encuentra es una lista. Para salvar este inconveniente se utiliza la función APPLY, que tiene el siguiente formato:

(APPLY ⟨función⟩ ⟨lista de argumentos⟩)

Si ahora se aplica dicha función

```
( APPLY '+ NUM )
14
```

Esta función se puede utilizar para operar con el resultado de la función MAPCAR, por ejemplo: se quiere calcular el número total de átomos que componen una expresión:

```
( DEFUN CONT-ATOM ( LISTA )
   ( COND    ( ( NULL LISTA ) 0 )
             ( ( ATOM LISTA ) 1 )
             ( T ( APPLY '+ ( MAPCAR 'CONT-ATOM LISTA ) ) )  )  )
```

— DO: es una primitiva más potente para la realización de iteraciones. La sintaxis de DO es un poco más larga, pero fácil de entender siguiendo un ejemplo:

$$a^b = a\,a\,a\,a\,a \ldots^{b} \ldots a$$

```
(DEFUN POTENCIA ( BASE EXP )
   ( DO ( ( CONT 1 )
          ( B EXP ) )
        ( ( ZEROP B ) RESULTADO )
        ( SETQ RESULTADO ( * RESULTADO BASE ) )
        ( SETQ B ( - B 1 ) ) )  )
```

Analizando la función, lo primero que se encuentra es una lista: ((CONT 1) (B EXP)) con dos expresiones, cada una contiene un paráme-

tro del bucle y su valor inicial. A continuación se encuentra otra lista compuesta por la condición de terminación del bucle: (ZEROP B) y la acción que se realizará:

RESULTADO. Por último, se encuentra una serie de acciones que se ejecutan en cada iteración del bucle. También se puede añadir a la lista de parámetros, una expresión que indica la forma de actualizar cada parámetro. La sintaxis completa es la siguiente:

(DO((⟨parametro$_1$⟩ ⟨inicialización$_1$⟩ ⟨actualización$_1$⟩)
(⟨parametro$_2$⟩ ⟨inicialización$_2$⟩ ⟨actualización$_2$⟩)

(⟨parámetro$_n$⟩ ⟨inicialización$_n$⟩ ⟨actualización$_N$⟩
⟨Cond de salida⟩ ⟨expresiones-intermedias⟩ ⟨expresión de salida⟩)
⟨Sentencias ejecutables en cada iteración⟩)

Todas las inicializaciones se realizan antes de realizar cualquier asignación y todas las actualizaciones se hacen antes de las reasignaciones. De esto se deriva que DO maneje sus parámetros en paralelo, es decir, que si un parámetro influye en otro a la hora de actualizar su valor, el valor que se va a utilizar para dicha operación es el que tenía en la iteración que acaba de realizarse.

Existe una variante de esta función que permite hacer la asignación y actualización de parámetros, se llama DO*. Antes de introducir la primitiva DO, la única manera de realizar operaciones iterativas era mediante la primitiva PROG. Al igual que DO, su sintaxis es bastante sencilla:

```
( DEFUN  EXPONENCIAR ( M N )
      ( PROG ( RESULTADO EXPONENTE )
            ( SETQ RESULTADO  1 )
            ( SETQ EXPONENTE  N )
      LOOP
            ( COND ( ( ZEROP EXPONENTE )
                  ( RETURN RESULTADO ) ) )
            ( SETQ RESULTADO ( * M RESULTADO ) )

      ( SETQ EXPONENTE ( - EXPONENTE 1 ) )
            ( GO LOOP ) ) )'
```

Lo primero que aparece a continuación de PROG es una lista que contiene los parámetros del bucle, estas variables son ligadas y se reasignan a su valor inicial después de salir de PROG. Si no existen parámetros, debe aparecer NIL o () en su lugar. A continuación, aparece la inicialización de los parámetros del bucle, y una etiqueta que delimita las funciones que se realizan en cada iteración del bucle. De ellas una tiene que aparecer ne-

cesariamente para que termine la ejecución de PROG: RETURN; en el momento que el intérprete encuentra dicha instrucción parará O la ejecución del bucle y devolverá como resultado el que indique la función. Para indicar la vuelta al principio del bucle, se utiliza la función (GO ⟨etiqueta⟩)

La función PROG admite anidamiento.

PROG1 y PROGN son dos variantes de esta primitiva; se utilizan para realizar secuencias de funciones devolviendo el valor de una de ellas.

```
( PROG 1 ( + 2 3 ) ( SETQ X 'Z) ( SETQ Y X ) )
6
( PROGN ( + 2 3 ) ( SETQ X 'Z ) ( SETQ Y X ) )
X
```

Lista de propiedades

Otro concepto que utiliza LISP es el de listas asociativas; consiste en una lista de listas, en las que el primer elemento es una clave de acceso a dicha sublista.

```
( SETQ COCHE-A '( ( COLOR VERDE )
      ( MARCA SEAT )
      (TIPO 127 ) ) )
( ( COLOR VERDE ) ( MARCA SEAT ) (TIPO 127 )) )
```

Para extraer el contenido de esta estructura se utiliza la primitiva ASSOC:

```
( ASSOC COLOR COCHE-A )
   ( COLOR VERDE )
( ASSOC MARCA COCHE-A )
   ( MARCA SEAT )
```

Hay que poner de manifiesto que la lista puede contener dos sublistas con la misma clave de acceso, ASSOC sólo devuelve el contenido de la primera lista que encuentre.

También se puede utilizar la primitiva GET para manejar las listas de propiedades, aunque esta primitiva es mucho más general.

```
( GET 'COCHE-A 'COLOR )
   VERDE
```

Si la propiedad no existe, devuelve NIL.

Las propiedades se pueden eliminar de las listas de propiedades mediante la primitiva REMPROP, que tiene la siguiente sintaxis:

(REMPROP ⟨nombre del símbolo⟩ ⟨nombre de la propiedad⟩)

Se puede utilizar la primitiva SETF para asignar o reasignar un valor a una propiedad.

```
( SETF ( GET 'COCHE-A'COLOR ) ROJO )
   ROJO
( GET 'COCHE-A'COLOR )
   ROJO
```

La primera expresión de SETF identifica el símbolo y la propiedad que se va a resignar. A continuación aparece el nuevo valor.

Definición de estructuras

Se pueden utilizar diversas estructuras como listas, listas de propiedades, estructuras creadas por cada programador, etc., dependiendo de cada circunstancia, pero es conveniente definir una estructura en el computador y utilizar muchas variables con esa misma estructura, sin necesidad de construirla cada vez. LISP proporciona una primitiva para definir estructuras similares a los registros: DEFSTRUCT. Se puede acceder individualmente a cada una de las partes de dichas definiciones.

```
( DEFSTRUCT ( PERSONA )
     ( SEXO        NIL )
     ( ALTURA      1.70 )
     ( PESO        65 )
     (PROF         NIL ) )
```

En primer lugar, se encuentra el nombre de la estructura entre paréntesis y, a continuación, una serie de listas compuestas por un par (campo, valor por defecto). La primitiva, ademas de definir la estructura, crea la función necesaria para asignar dicha estructura a una variable y para acceder a los distintos campos. En este caso, crea la función MAKE-PERSONA. Ahora se puede hacer una asignación de la siguiente forma:

```
( SETQ JUAN ( MAKE-PERSONA ) )
   #(PERSONA 71C6:12EA)
```

Si se quiere acceder al contenido de los distintos campos, se utilizarán las otras funciones que también crea DEFSTRUC:

```
( PERSONA-SEXO JUAN )
   NIL
( PERSONA-PROF JUAN )
   NIL
```

Si lo que se desea es modificar los valores de estos campos, se utilizará la función SETF:

```
( SETF ( PERSONA-PROF JUAN ) 'MEDICO )
   MEDICO
```



```
( PERSONA-PROF JUAN )
   MEDICO
```

También cabe la posibilidad de asignar valores distintos al crear una variable con la estructura definida:

```
( SETQ PEDRO (MAKE-PERSONA :PROF 'MILITAR :SEXO 'V :ESTATURA '1.80 : PESO '75 ) )
   #(PERSONA 798C6:196EA)
```

La utilización de esta primitiva es más apropiada que crear estructuras en base a combinaciones de listas y además ahorra trabajo, ya que no hay que hacer las funciones necesarias para acceder a los campos de la estructura.

Entrada/salida

Hasta el momento, la única manera de que el ordenador tome información es a partir de los argumentos de la función, y las únicas salidas son las proporcionadas por las funciones. A continuación se explica las primitivas básicas de E/S:

— PRINT evalúa su argumento y lo imprime en una línea aparte:

```
( DEFUN PARES ( N )
     ( DO (M 1 ( + M 1 ) )
          ( ( EQUAL M N ) ( PRINT ( * M 2 ) ) )
          ( PRINT ( * M 2 ) ) )  )
```

```
( PARES 4 )
   2
   4
   6
   8
```

A veces es necesario imprimir caracteres especiales que, como norma general, no están permitidos. Para salvar este inconveniente se pueden utilizar barras verticales o «\»:

```
( PRINT '|(| )
   |(|
( PRINT '\' )
   '
( PRINT '|UN BLANCO| )
   |UN BLANCO|
```

— READ: cuando el intérprete encuentra esta primitiva, se para y es-

pera a que el usuario introduzca un dato por el teclado. El valor que devuelve, es leído:

```
( READ ) UNO
    UNO
( SETQ EJEMPLO ( READ ) ) DOS
    DOS
```

Ya que READ nos indica que se está esperando un dato, es conveniente poner la función PRINT indicando que se quiere leer un dato.

— TERPRI: produce un salto de línea y coloca el cursor al principio de la línea.

— PRIN1: es igual que PRINT, con la diferencia de que no empieza la impresión en una nueva línea ni finaliza con un blanco.

```
( PRINT X ) = ( PROGN ( TERPRI ) ( PRIN1 X ) ( PRIN1 '| | ) )
```

— PRINC: igual que PRIN1, salvo que suprime las barras de los símbolos

```
( PRINC '|(| )
```

Matrices

LISP también permite la definición de matrices de una o dos dimensiones. Para crear una matriz se utiliza la primitiva MAKE-ARRAY:

```
( SETQ MATRIZ ( MAKE-ARRAY '(30 30 ) )
#( ARRAY 798C6:196EA )
```

Esta sentencia crea una matriz de nombre MATRIZ de dos dimensiones, cuyos índices varían entre 0 y 29. Por definición, los subíndices empiezan desde cero.

Para obtener el valor de un determinado elemento se utiliza la primitiva AREF:

```
( AREF MATRIZ 5 7 )
```

Si lo que se quiere es variar el contenido de uno de los elementos, se utilizará la primitiva SETF:

```
( SETF ( AREF MATRIZ 5 7 ) 80 )
    80
```

Si se desea saber el rango de unos de los subíndices de la matriz, se utiliza ARRAY-DIMENSION:

```
( ARRAY-DIMENSION MATRIZ 0 )
    30
```

La numeración de los subíndices también empieza en cero.

JUEGOS CONTRA LA NATURALEZA O DE UN SOLO JUGADOR 2

EL DISCO TONTO



L disco tonto consiste en cuatro círculos concéntricos, cada uno de los cuales puede girar alrededor de su centro. A su vez, cada círculo está dividido en ocho sectores iguales con un número escrito en cada uno. La figura 5 muestra la disposición del disco.

El objetivo del juego es girar los círculos del disco de forma que los números de cada radio o sector invisible sumen doce.

Este disco es un claro ejemplo de los juegos y problemas en los que el número de posibles combinaciones es relativamente grande. Es lo que se denomina explosión combinatoria. A este tipo de juegos también pertenece el ajedrez. Sin embargo, si se tienen en cuenta determinadas propiedades, el disco puede ser solucionado en unos pocos intentos.

Como puede observarse en la figura 5, si cada radio debe sumar 12, entonces cada diámetro debe sumar 24 y, por tanto, la cruz 48. Con lo cual, se puede establecer primero como meta el hacer que la cruz sume 48. Para lograr esto hay 24 posibles combinaciones diferentes. Los giros de los círculos de 90,° y 180,° no cambian la suma de los números de la cruz, pues la siguen componiendo los mismos números. Sin embargo, giros de +45° o de –45° cambian la configuración de la cruz. En cuanto se obtenga la submeta de que los números de la cruz sumen 48, se habrá reducido enormemente el problema.

La segunda submeta que ha de conseguirse es que los números de todos los diámetros sumen 24. Si se practicaran giros de +45° o –45° se quedaría rota la suma de la cruz. Por tanto, han de realizarse giros de 90°. Los giros de 180° no cambian la configuración, pues los diámetros siguen tomando los mismos números. El número de posibilidades es 24.

Por último, se ha de establecer la meta inicial, es decir, conseguir que los radios sumen 12. Sólo nos sirven giros de 180°, pues otro tipo de giro estropearía la configuración formada.

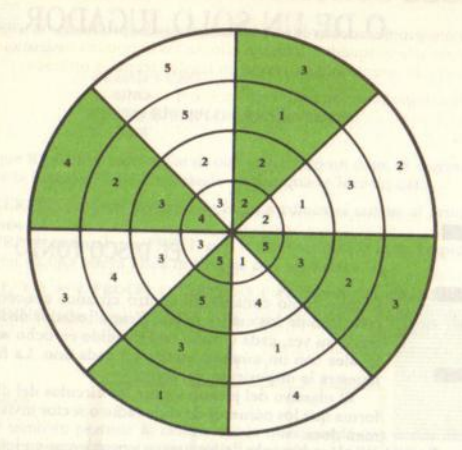


Fig. 5.

De esta manera, el número de ensayos necesarios para llegar a la solución es 72. Este número es muy inferior a las 8^4 posibles combinaciones. La solución a este problema, típico de la búsqueda heurística, se basa en dos principales características:

— Cada posición ganadora tiene una propiedad (cada radio suma 12). Esto es generalizable para casi todos los juegos y problemas de este libro. No sólo eso, sino que además esta propiedad envuelve o implica otras propiedades como, por ejemplo, la de que la suma de cada diámetro debe ser 24, y la suma de la cruz 48. Sin embargo, estas propiedades no implican una posición ganadora, a no ser que se tomen en conjunto.

— Existen determinados movimientos que no cambian el estado del problema. Hay que tenerlos en cuenta a la hora de obtener la solución, debido a que se pierde tiempo e intentos al realizarlos. Al mismo tiempo, se ha de tener presente que determinados movimientos pueden destrozar la situación alcanzada. Este es el caso, por ejemplo, de los giros de +/–45° cuando se ha conseguido ya que la cruz sume 48.

Estas dos propiedades ayudan a la resolución del problema restringiendo el espacio de búsqueda. Debido a ello, se las denomina heurísticas. La importancia de este problema reside en las dos técnicas utilizadas:

— Descomposición de problema en subproblemas.
— Asignación de heurísticas a cada subproblema.

EL 8 Y EL 15-PUZZLE

Estos dos problemas-juegos forman parte del conjunto de ejemplos más clásicos dentro del área de la IA dedicada a la heurística. El planteamiento del juego es muy sencillo:

Se posee un tablero de 3 × 3, dividido en 9 cuadrados. Ocho de estos cuadrados se pueden mover, mientras que el otro es un hueco. En los cuadrados móviles figuran números comprendidos entre el 1 y el 8. No puede haber dos cuadrados con el mismo número. Un posible estado del problema es el expresado en la figura 6.

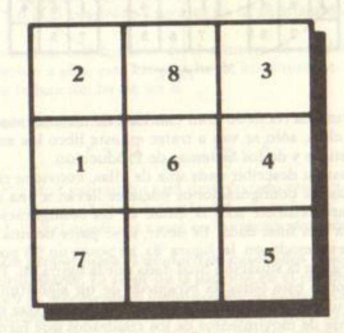


Fig. 6.

El juego consiste en transformar una situación inicial cualquiera, en la cual los números están desordenados, en una situación final que debe ser fijada antes de empezar. El juego se va desarrollando al mover el jugador las piezas o cuadrados hasta llegar a la situación final prevista.

Los movimientos de las piezas se realizan al pasar cualquier número vecino del cuadrado vacío a ocupar la posición de éste. Se consideran cua-

dros vecinos a aquellos que se encuentran inmediatamente arriba, debajo, al lado derecho y al izquierdo. En la figura 7 se pueden observar los movimiento posibles a partir de la configuración dada.

Por supuesto, si el cuadro vacío se encuentra en alguna de las esquinas, sólo existen dos posibles movimientos. Y si el cuadro vacío se encuentra en el centro, existen cuatro posibles movimientos.

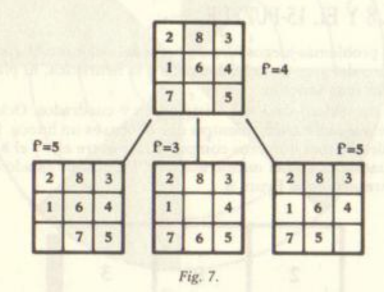


Fig. 7.

Este problema ha recibido gran cantidad de tratamientos dentro de la IA. Dentro de ellos, sólo se van a tratar en este libro los enfoques de las técnicas heurísticas y de los Sistemas de Producción.

Antes de pasar a describir cada una de ellas, conviene resaltar la idea de que no todas las configuraciones iniciales llevan a una configuración final dada. Aparentemente sólo la mitad de las configuraciones iniciales pueden llevar a una final dada. Es decir, si se parte de una situación inicial como la expresada en la figura 8a, se posee un 50 por 100 de posibilidades de llegar a la situación final dada por la figura 8b. Todavía se está investigando sobre este tema: la búsqueda de un algoritmo con el cual, partiendo de una situación inicial y otra final, se sepa que están conectadas por una serie de movimientos de los cuadrados que forman el puzzle. Sin embargo, como se explicará más adelante, en el 15-Puzzle sí existe dicho algoritmo.

Técnicas heurísticas

Si la técnica utilizada es la de escalada, lo primordial es definir la función de evaluación que ha de maximizarse o minimizarse. Este punto ha tenido varios enfoques.

2	8	3
1	6	4
7		5

Fig. 8a.

1	2	3
8		4
7	6	5

Fig. 8b.

El enfoque más sencillo es el que expresa la función de escalada de la siguiente forma:

$$f = \Sigma C_i, \qquad i = 1...8,$$

donde C_i es: * 1 si el cuadro i está colocado en la misma posición que en la que le corresponde del estado final.
* 0 si el cuadrado i no está en la misma posición.

Como se observa, esta función da el número de cuadrados colocados en su sitio. Debido a ello, esta función ha de maximizarse. En el estado final el valor de la función ha de ser 8.

Otra variante de esta función es:

$$f' = \Sigma D_i, \qquad i = 1...8,$$

donde D_i es: * 1 si el recuadro i está descolocado respecto de la posición que le corresponde del estado final.
* 0 si el cuadrado i está en la misma posición.

Esta función proporciona el número de cuadrados descolocados y, por ello, ha de minimizarse. El valor de la función en el estado final ha de ser 0. En la figura 9 se representa el camino seguido hasta obtener la solución mediante la utilización de la función f'. Se ha de tener en cuenta que en un determinado nodo no se desarrollan los estados visitados anteriormente en el camino de éste al nodo inicial. Si se hiciera esto, provocaría que los movimientos se desharían. Es decir, se podría permitir que después de mover un cuadro hacia arriba, en el siguiente movimiento se moviera hacia abajo, desandando el camino.

Esta función encuentra la solución, pero no siempre de forma óptima, más aun cuando no se puede realizar retroceso. El retroceso se puede incorporar si se desea. Si se quiere alcanzar la solución de forma óptima,

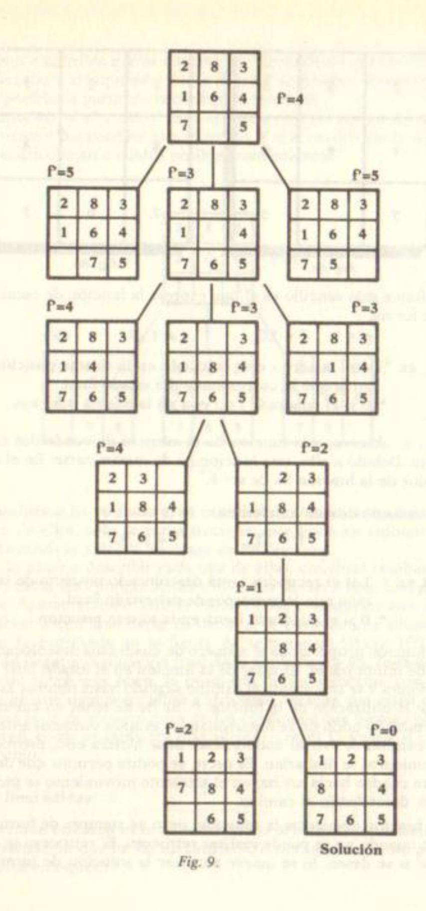


Fig. 9.

hay que recurrir a los métodos de búsqueda informada como el algoritmo A*. En concreto, f' representa el límite inferior de h(n). Recuérdese que h(n) es la función que estima la distancia que existe entre un nodo y el o los nodos meta. Es válida, por tanto, pero no es la mejor que puede utilizarse en el algoritmo A*. Nilsson (Nilsson, 1980) comenta otras funciones que mejoran en aproximación a f'. Como ejemplo cita la función:

$$h(n) = P(n) + 3 * S(n),$$

donde:

— P(n) es la suma de las distancias que existen entre las posiciones de los cuadrados en la configuración del nodo n y la posición de los mismos en la configuración final.

— S(n) es una función suma de los siguientes miembros:

- — se suma dos por cada cuadrado que no esté situado en la casilla central y no esté seguido por su sucesor correcto.
- — no se suma nada por cada cuadrado que no esté situado en la casilla central y esté seguido por su sucesor correcto.
- — se suma uno si existe un cuadrado en la casilla central y no es el vacío.

Sistemas de Producción

Se va a definir un posible Sistema de Producción que obtiene la solución. El Sistema de Producción va a estar formado por:

— Base de Hechos, constituida por dos elementos:

- — Una matriz que expresa la situación actual del «puzzle».
- — Un elemento que contiene la posición en la que se encuentra el cuadrado vacío en el instante considerado. Se expresará de la forma: vacío (fila, columna).

Esta información es redundante, pues se puede extraer de la matriz, pero facilita las operaciones del Sistema de Producción.

En la situación inicial, la Base dispondrá de la siguiente información para el ejemplo considerado:

Base de Hechos = <

2	8	3
1	6	4
7		5

, vacío (3, 2)>

Fig. 10

— Base de Reglas, que contendrá las siguientes reglas:

SI vacío(i,j) Y i $\neq$ 1 ENTONCES vacío(i–1,j)

SI vacío(i,j) Y i $\neq$ 3 ENTONCES vacío(i+1,j)

SI vacío(i,j) Y j $\neq$ 1 ENTONCES vacío(i,j-1)

SI vacío(i,j) Y j $\neq$ 3 ENTONCES vacío(i,j+1)

La primera regla dice que si el vacío está en la posición (i,j) y además no se encuentra en la primera fila, entonces puede moverse un cuadrado a la izquierda. Esta acción hará que en la Base de Hechos desaparezca el hecho vacío(i,j) y se sustituya por vacío(i-1,j). Al mismo tiempo, el sistema cambiará la configuración de la matriz.

Se puede observar que las cuatro reglas expresan los cuatro movimientos: arriba, abajo, izquierda y derecha. Está claro que en la mayoría de las ocasiones existirá conjunto conflicto que tendrá que resolver la Estrategia de Control.

— Estrategia de Control. En este caso se puede utilizar la mayoría de las técnicas estudiadas en el capítulo 4. Sin embargo, en este problema una de las mejores estrategias es realizar la resolución del conjunto conflicto mediante la aplicación del algoritmo A*. Para aplicar este algoritmo se puede utilizar cualquiera de las técnicas heurísticas anteriormente descritas en este capítulo. El funcionamiento del Sistema de Producción terminará cuando en la Base de Hechos aparezca la matriz correspondiente al estado final, o cuando no se pueda aplicar ninguna regla. Esto último indicaría que las dos configuraciones: inicial y final, no estaban conectadas.

Problema del «15 Puzzle»

Para este problema se puede aplicar cualquiera de las técnicas vistas en el apartado anterior. Pero no sólo eso, sino que, por medio de un algoritmo determinado, se puede saber si desde una posición o estado inicial dada se puede llegar a obtener una posición final dada. Esto define claramente el problema. Así, al comenzar la ejecución del programa que lo resuelva, se ejecutará dicho algoritmo. Si el resultado de la evaluación es correcto, se podrá continuar con el programa. En caso contrario, no se debe empezar, pues no se llegaría a la situación ganadora.

El algoritmo consiste en lo siguiente. Se calcula el número de cuadrados que no se encuentran en su posición final. Si este número es impar, el problema tiene solución. En caso contrario, no tiene solución.

LA FALSA MONEDA

La descripción del problema es la siguiente: se dispone de n monedas. Todas ellas pesan lo mismo menos una. Se trata de obtener el número medio mínimo de pesadas con el cual es posible determinar qué moneda es la que pesa diferente. Se dispone para ello de una balanza.

Para resolver el problema, se supondrá que la moneda que pesa diferente es más ligera que las otras. El suponer que es más pesada no supondría ninguna dificultad adicional, puesto que la solución es análoga a la que se tratará en este capítulo. Se ha logrado demostrar que el mínimo número de pesadas necesarias para llegar a determinar la moneda falsa con estas condiciones es min $(k/n \leq 3^k)$. Un problema similar se plantea cuando no se sabe si la moneda de peso diferente es más pesada o más ligera.

Este problema va a resolverse utilizando la técnica de escalada. Para ello se necesita una función de evaluación que hay que maximizar o minimizar. En concreto, una de las funciones de evaluación que mejores resultados da para este problema es la siguiente:

$$f = N_n * P_n + N_d * P_d$$

donde:

— N_n es el número de monedas cuyo peso es desconocido después de efectuar una pesada, dado que la balanza se ha nivelado.
— N_d es el número de monedas cuyo peso es desconocido después de efectuar una pesada, dado que la balanza se ha desnivelado.
— P_n es la probabilidad de que la balanza se nivele.
— P_d es la probabilidad de que la balanza se desnivele.

Por tanto, esta función expresa el número medio o esperado de monedas desconocidas después de efectuar una pesada. Esto implica que habrá que minimizarla.

A continuación se pondrá un ejemplo para clarificar el tema. Se parte de 24 monedas de las que una moneda pesa menos que las demás. Bajo estas condiciones esta función de escalada sí llega a obtener el número medio mínimo y será la profundidad del árbol que se construirá.

Si se colocan diferentes números de monedas en los brazos de la balanza, no se obtendrá ningún resultado importante o por lo menos existen pocas probabilidades de obtenerlo. Por tanto, se han de pesar un número par de monedas en cada pesada, de forma que en cada plato de la balanza se disponga de igual número de monedas. Si, por ejemplo, se pesan 12 monedas con las otras 12, se extraerá la siguiente información. El número de monedas cuyo peso es desconocido después de la pesada es 12. Si la balanza se inclina hacia la derecha, se sabe que ahí no está la mone-

da falsa, pues ésta ha de estar en el montón que menos pese. Con lo cual, el número de monedas cuyo peso es desconocido es 12 (las del montón que menos pesa). No puede suceder que la balanza se nivele, puesto que por hipótesis, tiene que haber una moneda que pese menos.

La función de evaluación en este caso valdría:

$$f = 0 + 12 * 1 = 12,$$

siendo: $P_n = 0$
$N_d = 12$
$P_d = 1$

Supóngase a continuación que se pesan 11 monedas con otras 11. Los resultados de esta pesada serían:

— Número de monedas desconocidas si la balanza se nivelase = 2, puesto que la moneda falsa estaría entre las dos que no se pesaron.

— Número de monedas desconocidas si la balanza se desnivelase = 11, puesto que la moneda falsa estaría en el montón que menos pesara y se sabría el peso de todas las demás.

— Probabilidad de que la balanza se nivele = 2/24. Es la misma que la probabilidad de que la moneda falsa sea una de las dos que no se pesaron.

— Probabilidad de que la balanza se desnivele = 22/24. Es la misma que la probabilidad de que la moneda falsa sea una de las 22 que se pesaron.

En este caso, la función de evaluación tomaría el valor:

$$f = 2 * 2/24 + 11 * 22/24 = 10.25.$$

Como 12 < 10.25, será mejor, para la resolución del problema pesar 11 con 11 que 12 con 12.

Si se siguiera calculando esta función para las pesadas de 10 con 10, 9 con 9, 8 con 8, etc., se obtendrían los valores necesarios para formar el primer nivel del árbol de búsqueda dado en la figura 11.

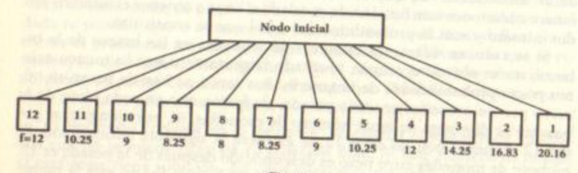


Fig. 11.

A continuación, según la técnica de escalada, se seguirá explorando por la rama que posea la función de escalada menor. En este caso es el 8. Después de efectuar el mismo proceso de pesadas, el valor menor de la función lo da la rama de las 3 monedas. Por último, se puede comprobar que en la siguiente pesada se localizará la moneda falsa. Esto es debido a que si pesamos tres monedas con otras tres pueden suceder dos casos:

— La balanza se equilibra con lo cual sólo se desconocerá el peso de dos monedas. Esto hace que en la próxima pesada se sepa cuál es la falsa moneda.

— La balanza se desequilibra por lo que habrá tres monedas cuyo peso sea desconocido. Con pesar 1 con 1 en la siguiente pesada, se sabrá cuál es la moneda falsa.

Al final del proceso se obtendrá el árbol de la figura 2. La solución en este caso es 3 pesadas como número medio mínimo de pesadas para descubrir la moneda falsa.

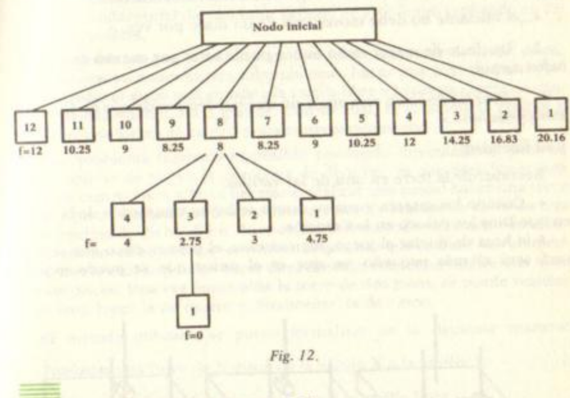


Fig. 12.

LAS TORRES DE HANOI

El juego de las Torres de Hanoi o de los discos está tomado de la tradición indoasiática:

«En el gran templo de Benarés, bajo el domo que marca el centro del mundo, yace una placa de latón a la que están fijadas tres agujas de dia-

mante de un codo de alto cada una, y del grosor del cuerpo de una abeja. En torno a estas agujas, Dios, en el acto de la Creación, colocó sesenta y cuatro discos de oro puro, descansando el mayor sobre la placa de latón y decreciendo progresivamente los demás conforme se asciende por la pila. Es la torre de Brahma. Día y Noche, sin cesar, los sacerdotes del templo van transfiriendo los discos de una aguja de diamante a otra, de acuerdo con las leyes fijas e inmutables de Brahma, que exigen que el oficiante no deba mover más de un disco por vez y que deba situar tal disco en otra aguja sin que quede por debajo de él discos más pequeños. Cuando los sesenta y cuatro discos se hayan transferido de la aguja en que Dios los colocó en la Creación a una de las otras, la torre, el templo y los Brahamines quedarán reducidos a polvo y, con inmenso estruendo, el mundo se desvanecerá.»

Dejando aparte la veracidad de la leyenda, del texto se pueden extraer las reglas básicas que rigen el juego:

1. Sólo se puede trasladar un disco en cada movimiento:

«...el oficiante no debe mover más de un disco por vez...»

2. Un disco de mayor radio nunca puede estar por encima de otro de radio menor:

«... situar tal disco en otra aguja sin que queden por debajo de él discos más pequeños...»

y su finalidad:

Reconstruir la torre en otra de las varillas:

«...Cuando los sesenta y cuatro discos se hayan transferido de la aguja en que Dios los colocó en la Creación...»

A la hora de iniciar el juego, lógicamente, el primer disco que se moverá será el más pequeño, ya que es el único que se puede mover.



CASO CON TRES PISOS

En el siguiente paso se presentan dos posibilidades: mover el disco pequeño a la otra varilla libre (poco aconsejable) o mover el segundo disco desde la torre a la varilla libre (que es más lógico). Es evidente que no se podrá colocar sobre la varilla ocupada por el disco más pequeño, porque se violaría la regla número dos.

A partir de este paso, no es tan obvio lo que se tiene que hacer, ya que los posibles movimientos aumentan y sólo alguno de ellos lleva directamente a la solución. Aun si en todos los casos se escogiese el movimiento correcto, se necesitarían $2^n - 1$ para alcanzar la solución, siendo n el número de discos que componen la torre.

Después de jugar un rato con la torre de cinco pisos se puede observar que aparecen de tiempo en tiempo torres de dos, tres o cuatro pisos y se puede detectar una cierta regularidad en la formación de estas torres. Estas pautas pueden llevar a descubrir el mecanismo de resolución del juego.

El método que se va a utilizar está basado en la descomposición de problemas irresolubles en subproblemas que sí pueden resolverse. Esta idea es la base fundamental del resolutor general de problemas explicado en capítulos anteriores.

Si el problema que se está tratando es hacer una torre de cinco pisos, se puede descomponer en tres subproblemas: hacer una torre de cuatro pisos, trasladar el disco más grande a la varilla libre y hacer una torre de cuatro pisos encima del disco grande. Con lo cual, la solución se reduce a construir dos torres de cuatro pisos convenientemente.

Este problema tampoco es posible resolverlo directamente, pero se puede aplicar de nuevo la estrategia de división, ya que para hacer una torre de cuatro pisos sólo es necesario realizar tres pasos: hacer una torre de tres pisos, trasladar el disco número cuatro y realizar la torre de tres pisos encima de dicho disco. Si se aplica este método reiteradamente llega un momento en que los movimientos que hay que realizar son elementales, con lo cual se pueden hacer directamente, cuando se realice una torre de dos discos. Una vez construida la torre de dos pisos, se puede realizar la de tres, luego la de cuatro y, finalmente, la de cinco.

El método utilizado se puede formalizar de la siguiente manera:

Trasladar una torre de N pisos de la varilla X a la varilla Y:

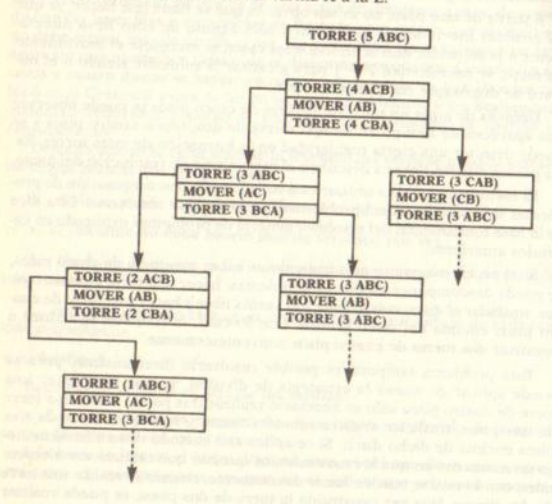
1. Trasladar los N-1 pisos superiores a la varilla libre.
2. Trasladar el disco más grande a la varilla Y.
3. Trasladar los N-1 discos superiores desde la varilla libre hasta la varilla Y.

Este procedimiento es recursivo, ya que en la definición de la solución

interviene la propia solución. Falta establecer la condición de parada de las llamadas recursivas:

Si la torre es de un solo piso:

1. Trasladar el disco desde la varilla X a la Z.



Esto desarrollado en LISP sería:

```
( DEFUN TORRE ( ALTURA ORIGEN DESTINO AUX )
            ( COND ( ( NOT ( ZEROP ALTURA ) )
                   ( TORRE ( - ALTURA 1 ) ORIGEN AUX DESTINO )
                   ( MOVER ORIGEN DESTINO )
                   ( TORRE ( - ALTURA 1 ) AUX DESTINO ORIGEN ) )
            ( T ( MOVER ORIGEN DESTINO ) ) ) )
```

donce MOVER realizaría la operación de pasar el primer disco desde ORIGEN hasta DESTINO.

Quizá el procedimiento recursivo sea complicado, pero no importa, también existe una solución iterativa al problema:

1. Trasladar el disco pequeño a la varilla consecutiva en el sentido de las agujas del reloj.
2. Traslación de cualquier disco, a excepción del mínimo.

Este segundo paso, aunque parece poco restrictivo, en el fondo sí que lo es, ya que sólo una pieza cumple la condición en cada momento, y sólo una varilla donde mover dicha pieza, ya que la otra tiene en su cima el disco pequeño.



LABERINTO

La mitología griega relata que el héroe Teseo entró a un laberinto para hallar y matar al monstruo Minotauro. Le ayudó Ariadna, quien le dio un ovillo de hilo. Teseo fue desenredándolo conforme se adentraba en el laberinto, mientras Ariadna sujetaba uno de los extremos del hilo. Después, al enredar de nuevo el hilo, halló fácilmente la salida.

Se relata esta antigua leyenda por su semejanza con un invento llamado «el ratón en el laberinto», que hizo el matemático e ingeniero americano Claude Shannon. El «ratón» se coloca en una posición dentro de un laberinto especial y otro objeto, que podría llamarse un «trozo de queso», se coloca en otra posición. El ratón anda por el laberinto siguiendo un camino tortuoso hasta que encuentra el «queso», pero después, si se coloca de nuevo en la misma posición de partida, se dirigirá directamente hacia el «queso» sin desvío alguno.

Se va a considerar un problema relacionado con la búsqueda de un camino a través de un laberinto (o, más correctamente, una clase de tales problemas) y se desarrolla un procedimiento que resuelve esta clase de problemas.

Se puede pensar en el laberinto como un sistema finito de uniones, de las cuales emanan los corredores. Cada corredor enlaza dos uniones (que se llaman adyacentes). También pueden existir uniones del tipo «callejón sin salida». De los cuales parte un solo corredor. El laberinto puede representarse geométricamente por un grafo donde los corredores están representados por arcos y los cruces por nodos.

Para construir el procedimiento, se usa un método especial de búsqueda. En cada paso de la búsqueda pueden separarse los corredores en tres clases: aquellos por los que Teseo nunca ha pasado (libres), aquellos por los que ha pasado una vez (semilibres) y aquellos por los que ha pasado

dos veces (cerrados). Además, desde cualquier unión, Teseo puede moverse hacia una unión adyacente de una de las dos maneras siguientes:

— desenredando el hilo, Teseo se mueve a lo largo de cualquier corredor libre hasta una unión adyacente, desenredando el ovillo de Ariadna conforme avanza; después, este corredor se considera como semilibre.

— «reenredando» el hilo, Teseo regresa a lo largo de un corredor semilibre hasta una unión adyacente, enredando el hilo conforme avanza; después, este corredor se considera cerrado.

Se supone que Teseo no avanzará por los corredores que estén cerrados, y que puede señalar en qué estado se encuentra cada corredor. La elección de cada movimiento depende de las condiciones que encuentra Teseo al llegar a una unión. Estas condiciones pueden ser algunas de las siguientes:

1. Que encuentre el Minotauro.
2. Que encuentre el hilo. Esto indica que ya ha pasado por esa unión.
3. Que haya por lo menos un corredor libre que parte de la unión.
4. Que se encuentre con Ariadna.
5. Que ocurra cualquier otra cosa.

A partir de estas cinco condiciones se puede describir el método de búsqueda:

SI	1	ENTONCES	Detenerse.
SI	2	ENTONCES	Enrollar el hilo en el ovillo hasta la unión anterior.
SI	3	ENTONCES	Seguir por el corredor libre.
SI	4	ENTONCES	Detenerse.
SI	5	ENTONCES	Enrollar el hilo en el ovillo hasta la unión anterior.

Al encontrarse en una unión cualquiera, Teseo decide su próximo movimiento de la siguiente forma: ejecuta la primera condición que se cumpla de acuerdo al orden establecido en las reglas.

Esto se puede realizar mediante un sencillo Sistema de Producción, donde la base de hechos estará compuesta por el grafo y las condiciones que se den en cada nodo, así como la posición de los tres personajes. La base de reglas es la expuesta anteriormente. Y la estrategia de control consiste en aplicar la primera regla que se cumpla.

MISIONEROS Y CANIBALES

En la orilla izquierda de un río hay *n* misioneros y *n* caníbales. Los misioneros quieren pasar a los caníbales a la otra orilla, para lo que disponen de una barca que sólo tiene capacidad para dos personas. En ninguna orilla puede ser mayor el número de misioneros que de caníbales, puesto que de no ser así los caníbales se comerían a los misioneros.

Va a realizarse la resolución del problema en base a un Sistema de Producción. El estado del problema después de cada movimiento puede expresarse en forma de lista. Esta lista podría representarse como sigue:

(MI, CI, MB, CB, ORILLA)

donde se incluyen el número de misioneros y caníbales en la orilla izquierda (MI y CI), así como los misioneros y caníbales en la barca (MB y CB) y la orilla en que se encuentra la barca (ORILLA).

En cada momento pueden calcularse fácilmente el número de caníbales y misioneros en la otra orilla. El problema consiste en pasar del estado inicial (3, 3, 0, 0, Izq) al estado (0, 0, 0, 0, Der) en el que los caníbales, misioneros y barca están en la orilla derecha.

Para pasar de una orilla a otra son necesarios una serie de movimientos que pueden expresarse en forma de reglas de producción. Algunas de las reglas correspondientes a cada movimiento son:

* Subir un misionero a la barca:

(Izq) $(MI - 1 \geq CI)$ $(MB + CB + 1 \leq 2) \rightarrow (MI - 1, CI, MB + 1, CB, Izq)$

Si la barca está en la orilla izquierda, quedan más misioneros que caníbales en la orilla izquierda y la barca no está aún llena, entonces subir un misionero a la barca.

* Subir un caníbal a la barca:

(Izq) $(CI > 0)$ $(MB + CB + 1 \leq 2) \rightarrow (MI, CI - 1, MB, CB + 1, Izq)$

Si la barca está en la orilla izquierda, hay algún caníbal en la orilla izquierda y la barca no está aún llena, entonces subir un caníbal a la barca.

* Cruzar el río:

(Izq) $(MB + CB > 0) \rightarrow (MI, CI, MB, CB, Der)$

Si la barca está en la orilla izquierda y no está vacía, entonces pasar a la orilla derecha.

(DER) $(MB + CB > 0) \rightarrow (MI, CI, MB, CB, Izq)$.

Si la barca está en la orilla derecha y no está vacía, entonces pasar a la orilla izquierda.

* Bajarse en el lado derecho:

(Der) $(3 - (MI + MB) \geq 3 - (CI + CB) + 1)$ $(3 - (MI + MB) > 0) \rightarrow (MI, CI, MB, CB - 1, Der)$

Si la barca está en el lado derecho, hay un caníbal en la barca y hay suficientes misioneros en la orilla derecha, entonces bajar un caníbal.

(Der) $(3 - (MI + MB) = 0)$ $(CB > 0) \rightarrow (MI, CI, MB, CB - 1, Der)$

Si la barca está en el lado derecho, hay un caníbal en la barca y no hay ningún misionero en la orilla derecha, entonces bajar un caníbal.

(Der) (MB – 1 $\geq$ CB) (MB > 0) $\rightarrow$ (MI, CI, MB – 1, CB, Der)

Si la barca está en el lado derecho, hay misioneros en la barca y al bajarse un misionero quedara mayor o igual número de misioneros que caníbales, entonces bajar un misionero de la barca.

La condición de finalización viene expresada como:

(Der) (3 – (MB + MI) = 3) (3 – (CB + CI) = 3) $\rightarrow$ FIN

Si están la barca, los tres misioneros y los tres caníbales en la orilla derecha, entonces se ha llegado a la solución.

La Base de Hechos inicialmente estará constituida por los estados inicial y final. La Base de Reglas contiene las reglas correspondientes a los movimientos.

Para la resolución del conjunto conflicto puede utilizarse como estrategia de control la técnica de búsqueda en amplitud. A lo largo del proceso se incluirán en la Base de Hechos los estados a los que se pasa después de aplicar las reglas, comprobando si coinciden con el estado final incluido en la Base de Hechos. Si esto fuera así, el procedimiento concluye describiendo el camino existente desde el nodo meta hasta el nodo inicial a través de todos sus antecesores. Dicha descripción puede llevarse a cabo estableciendo un puntero desde cada nodo a su nodo antecesor.

Un camino que lleva a la solución es:

Fig. 13 a.

Fig. 13 b.

A continuación se expone una forma sencilla de programar este problema. Como ya se indicó en la notación de las reglas, cada estado del problema se representará por una lista. En la Base de Hechos irán introduciéndose sucesivamente todos los estados por los que pasa la resolución del problema.

El procedimiento que crea la Base de Hechos inicial podría ser:

```
( DEFUN COMIENZO NIL
        ( SETQ MI 3 )
        ( SETQ CI 3 )
        ( SETQ MB 0 )
        ( SETQ CB 0 )
        ( SETQ ORI 'Izq )
        ( SETQ BH ( LIST ( LIST MI CI MB CB ORI ) ) )
```

En cuanto a las reglas, tendrán dos partes diferenciadas: una parte antecedente de condición de aplicación de la regla y otra consecuente de acción.

Para determinar si una regla es aplicable o no, se establecerá una función «booleana» que compruebe la condición y devuelva T o NIL, dependiendo de si la regla es aplicable o no.

Por ejemplo, para la regla que cruza el río de la orilla derecha a la izquierda, dicha función lógica sería COND3:

```
( DEFUN  COND3  ( L )
( AND  ( EQUAL ( CAR ( CDDDDR L ) )  'Der )
          ( GREATERP ( PLUS ( CADDR L ) ( CADDDR L ) )  0 ) ) )
```

Si esta regla se aplicara, se llamará a una función que modifica el estado actual. Para la regla anterior sería:

```
( DEFUN  ACCION3 NIL
        ( SETQ ORI 'Izq ) )
```

La forma más sencilla de realizar las inferencias sería comprobar sucesivamente cada una de las condiciones (COND1, COND2, ...) de las reglas, aplicando aquella acción (ACCION1, ACCION2, ...) que corresponda a la primera condición que se ha satisfecho. Habrá que comprobar, en primer lugar, si ya se ha alcanzado el estado final (0, 0, 0, 0, Der) y si no las condiciones de las reglas ordenadas de mayor a menor orden de prioridad. Todo esto se puede realizar con la función MISCAN.

```
( DEFUN MISCAN NIL
   ( SETQ  BOOL T )
   ( COND
        ( ( CONDFIN ( CAR BH ) )  BH )
        ( T ( COND
              ( ( COND1 ( CAR BH ) ) ( ACCION1 ) )
              ( ( COND2 ( CAR BH ) ) ( ACCION2 ) )
              ( ( COND3 ( CAR BH ) ) ( ACCION3 ) )
                 ................................
                 ................................
              ( ( CONDn ( CAR BH ) )  ( ACCIONn ) ) )
              ( T  ( PRINT '( NO  SE  PUEDE  APLICAR  NINGUNA
              REGLA ) )
                 ( SETQ BOOL NIL ) ) )
              ( COND ( BOOL
                     ( SETQ BH ( CONS ( LIST MI CI MB CB ORI ) ) )
                     ( MISCAN ) ) ) ) ) )
```

Se sugiere al lector que realice las modificaciones oportunas si quiere que la resolución del conjunto conflicto sea mediante búsqueda en amplitud.



EL JUEGO DE LAS OCHO REINAS

El juego o problema de las ocho reinas constituye un ejemplo muy conocido de la utilización de métodos de programación de tanteo sistemático, es decir, de prueba y error. También es un buen ejemplo de la utilización de los procedimientos de vuelta atrás o retroceso.

Este juego fue investigado por C. F. Gauss en 1850, aunque no llegó a resolverlo completamente, ya que este problema se caracteriza por su escasa adecuación a una solución analítica. La aparición de los ordenadores ha hecho posible la resolución del problema de forma mucho más sencilla.

El planteamiento del juego es el siguiente: hay que situar ocho reinas en un tablero de ajedrez de forma que ninguna de ellas pueda actuar sobre cualquiera de las otras con arreglo a las restricciones de movimientos de las reinas en el ajedrez. Es decir, es necesario tener en cuenta que, de acuerdo con estas reglas, una reina actúa sobre todas las piezas situadas en la misma columna, fila o diagonal del tablero. De ello se deduce inmediatamente que cada columna, fila o diagonal puede contener una única reina.

Una forma de resolver el problema consiste en ir colocando las reinas de una en una, fila por fila, en las columnas que vayan quedando libres de ataque. De esta forma las posibilidades de colocación de cada reina van disminuyendo según se van situando más sobre el tablero. Esta estrategia no conduciría a la realización de un buen programa, ya que en el momento en que alguna configuración parcial del problema viole cualquiera de las condiciones, no puede rectificarse añadiendo más reinas al tablero, con lo que el juego se bloquea casi cuando acaba de empezar.

Se puede seguir, por el contrario, una heurística que, en cada situación, permita decidir la acción que parece más ventajosa con arreglo a las características del problema. Una buena heurística en este caso colocaría cada reina dejando libres tantas opciones como sea posible para colocar las reinas posteriores. Así, pues, es necesario tener en cuenta el número de cuadros de la filas vacías que no son atacados por las reinas ya colocadas. De esta forma se tendría una medida de la bondad de una situación determinada mediante la función f(a) que calcula el número de cuadros no atacados si se coloca una reina en la posición a. Calculando la función f para todos los cuadros de una fila y eligiendo aquel que da resultado mayor, se puede implementar esta heurística en un programa. Existen también otras heurísticas que pueden representarse de forma parecida.

La implementación que resuelve el juego ha sido típicamente un procedimiento de búsqueda con vuelta atrás. Existen muchas versiones del juego en casi todos los lenguajes de programación de alto nivel. Algunos de ellos obtienen una única solución, mientras que otros obtienen todas las posibles. También es posible la programación del juego en LISP, lenguaje propio de la IA. El programa tiene la siguiente forma:

```
( DEFUN REINAS ( TAMAÑO )
     ( PROG ( N M TABLERO )
          ( SETQ N 1 )                                   ; 1ª fila
          LOOP-N
          ( SETQ M 1 )                                   ;1ª columna
          LOOP-M
          ( COND ( ( CONFLICTO N M ) ( GO DESHACER-M ) ) )
          ( SETQ TABLERO ( CONS ( LIST N M ) TABLERO ) )
          ( COND ( ( GREATERP ( SETQ N ( ADD1 N ) ) TAMAÑO )
               ( PRINT ( REVERSE ( TABLERO ) ) ) ) )
          ( GO LOOP-N )
          DESHACER-N
          ( COND ( ( NULL TABLERO ) ( RETURN 'TERMINADO ) )
               ( T ( SETQ M (CADAR TABLERO ) )
                 ( SETQ M ( CAAR TABLERO ) )
```

```
            ( SETQ TABLERO ( CDR TABLERO ) ) ) )
        DESHACER-M
        ( COND ( ( GREATERP ( SETQ M ( ADD1 M ) ) TAMAÑO )
                ( GO DESHACER-N )
              ( T ( GO DESHACER-M ) ) ) ) )
```

El programa no resuelve únicamente el problema de las ocho reinas, ya que admite la especificación del tamaño del tablero, con lo que puede extenderse el juego a un tablero mucho mayor. La estructura del programa es la de vuelta atrás. Hace un movimiento, y si no encuentra ningún problema, continúa colocando la siguiente reina. Cuando encuentra algún problema, por ejemplo que no puede seguir, deshace los movimientos realizados y vuelve a intentarlo. En este caso, la representación del tablero se hace mediante una lista que contiene pares de números que indican la fila y la columna en la que se ha colocado una reina. La función que decide si existe algún problema en la colocación de una reina en una determinada posición se denomina CONFLICTO. Su estructura es la siguiente:

```
( DEFUN CONFLICTO ( N M TABLERO )
    ( COND ( ( NULL TABLERO ) NIL )
    ( ( OR ( AMENAZA N M ( CAAR TABLERO ) ( CADAR TABLERO ) )
          ( CONFLICTO N M ( CDR TABLERO ) ) ) ) ) )

( DEFUN AMENAZA ( I J A B )
    ( OR (= I A )                         ;Igual fila
         ( = J B )                        ;Igual columna
         ( = ( - I J ) ( - A B ) )        ;Diagonal SO-NE
         ( = ( + I J ) ( + A B ) ) ) ) )  ;Diagonal NO-SE
```

La función CONFLICTO utiliza otra función auxiliar, AMENAZA, encargada de comprobar si la posición elegida se encuentra amenazada por alguna de las reinas ya colocadas en el tablero.

Cuando el programa termina, devuelve la lista TABLERO con las posiciones en las que están colocadas las reinas, y después escribe la palabra TERMINADO. Si se desea una respuesta más gráfica, pueden utilizarse las siguientes funciones que dibujan el tablero y las reinas situadas sobre él:

```
( DEFUN DIBUJA-TABLERO ( TABLERO )
    ( PROG ( TAMAÑO )
          ( SETQ TAMAÑO ( LENGTH TABLERO ) )
          ( DIBUJA-TABLERO-AUX TABLERO ) ) )
```



```
( DEFUN DIBUJA-TABLERO-AUX ( TABLERO )
    ( TERPRI )
    ( COND ( ( NULL TABLERO ) )
          ( T ( DIBUJA-TABLERO-SUB ( CADAR TABLERO ) 1 )
              ( DIBUJA-TABLERO-AUX (CDR TABLERO ) ) ) ) ) )

( DEFUN DIBUJA-TABLERO-SUB ( COLUMNA N )
    ( COND ( ( GREATERP N TAMAÑO ) )
          ( T ( COND ( ( EQUAL COLUMNA N ) ( PRINC 'Q ) )
                    ( T (PRINC '| . | ) ) ) )
              ( PRINC '| | )
              ( DIBUJA-TABLERO-SUB COLUMNA ( ADD1 N ) ) ) ) ) )
```

También es posible implementar el programa en forma recursiva en lugar de utilizar la vuelta atrás.

El juego de las ocho reinas es fácilmente representable y resoluble mediante un sistema de producción. Para la representación del problema han de tenerse en cuenta las diferentes filas, colocando una reina en cada una de ellas, guiándose por las siguientes reglas:

Si I = 1 entonces (poner reina en fila 1 columna 1) (I = 2)
Si I = 1 entonces (poner reina en fila 1 columna 2) (I = 2)
Si I = 1 entonces (poner reina en fila 1 columna 3) (I = 2)
Si I = 1 entonces (poner reina en fila 1 columna 4) (I = 2)

....

Si I = 2 entonces (poner reina en fila 2 columna 1) (I = 3)
Si I = 2 entonces (poner reina en fila 2 columna 2) (I = 3)

....

Si I = 8 entonces (poner reina en fila 8 columna 8) (I = 8)

El efecto de cada una de las reglas es la colocación de una reina en una determinada posición y el paso a la fila siguiente para examinar las posiciones libres. El índice 1 es el que corresponde a la fila que se examina en ese momento. Como puede observarse, en cada momento pueden aplicarse varias reglas. Para resolver el conflicto y elegir sólo una de ellas se hace uso de una función que evalúa la conveniencia de cada posición. La función se define de la siguiente forma:

diag(i, j)=longitud de la diagonal más larga que pasa por la casilla (i,j)

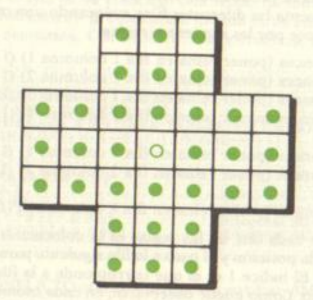
Como ya se indicó en la descripción del juego, una buena estrategia consiste en la elección de la posición que deje el mayor número de casillas sin atacar. La función descrita ilustra una manera de cuantificar tal estrategia.

La resolución del conflicto, por tanto, se hará eligiendo la regla cuya aplicación (colocación de una reina en una casilla concreta) produzca un valor menor de la función DIAG. En el caso de que existan dos valores mí-

nimos con dos reglas diferentes, se aplicará aquélla que coloque la reina en la columna situada más a la izquierda. De igual forma podría elegirse el criterio contrario, y colocar la reina en la columna situada más a la derecha.

SOLITARIO

Se juega sobre un tablero de 33 agujeros donde al principio del juego todos los agujeros menos el central están ocupados por clavos; en la figura puede verse cómo están dispuestos los agujeros y cuál es la situación inicial. Un movimiento consiste en hacer saltar un clavo sobre el adyacente, situándolo en un agujero vacío. El clavo sobre el que se saltó se extrae del tablero, pasando a estar el agujero vacío. El objetivo final es alcanzar un estado en el que en el tablero sólo haya un clavo en el agujero central.



El problema presenta una serie de características que ayudan a su resolución.

En cualquier momento, el número de pasos para alcanzar este estado es uno menos que el número de clavos situados en el tablero. Por tanto, habrá que realizar 31 pasos para llegar a la solución. Si quiere utilizarse una estrategia basada en un árbol de búsqueda, las posibles combinaciones para un nivel 31 son muy elevadas, lo que obliga a introducir poda, según alguna función heurística.

Profundizando en la geometría del problema, puede observarse que un clavo determinado sólo puede moverse dos posiciones a lo largo de los ejes horizontal y vertical; así se determinan cuatro tipos de agujeros, según se muestra en la figura 13, de tal forma que un clavo en un agujero sombreado sólo puede moverse a otro con el mismo sombreado.

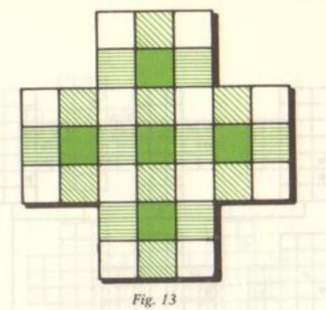
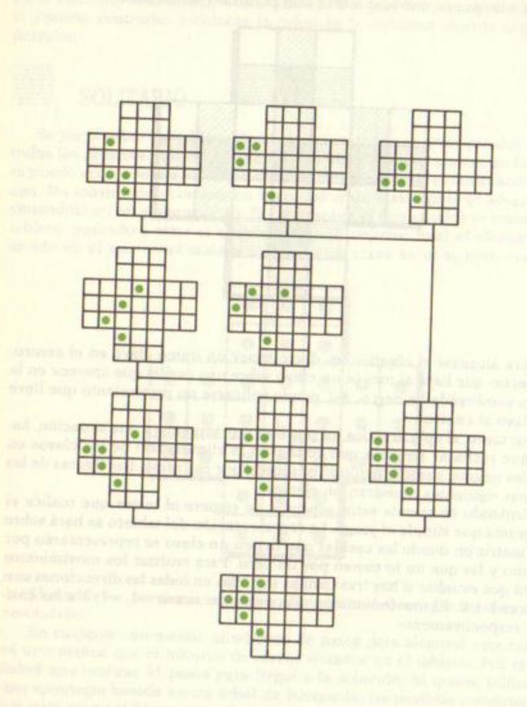


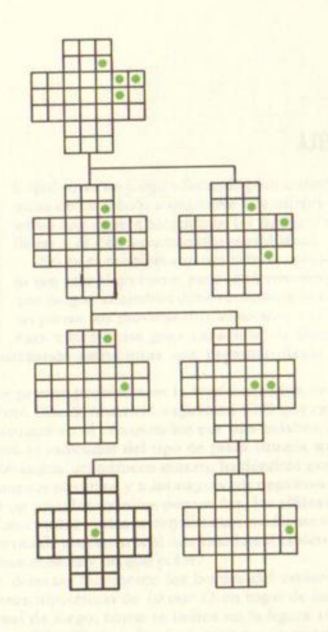
Fig. 13

Para alcanzar el objetivo, es decir, tener un único clavo en el centro, es preciso que haya al menos un clavo sobre una casilla que aparece en la figura sombreada de negro. Así, puede aplicarse un movimiento que lleve un clavo al centro.

Por tanto, al aplicar todos los posibles movimientos a una situación, habrá que rechazar aquéllos que conllevan la eliminación de los clavos en casillas negras, suspendiendo la búsqueda por esa rama. Las figuras de las páginas siguientes muestran un ejemplo.

Teniendo en cuenta estos aspectos, se sugiere al lector que realice el programa que simule el juego. La representación del tablero se hará sobre una matriz en donde las casillas que tienen un clavo se representarán por un uno y las que no lo tienen por un cero. Para realizar los movimientos habrá que estudiar si hay tres casillas seguidas en todas las direcciones con valores 1 1 0. El movimiento en sí consiste en sumar −1, −1, 1 a las casillas, respectivamente.





JUEGOS CONTRA UN CONTRINCANTE INTELIGENTE 3

AJEDREZ



EL ajedrez es un juego adecuado para tratarlo mediante técnicas de IA debido a que tiene claramente definidos el objetivo que quiere alcanzarse (el mate) y los medios para llegar a él (los movimientos permitidos).

No es el problema ni tan simple como para ser trivial, ni tan complejo como para ser irresoluble. Un programa que juegue al ajedrez debe reconocer la configuración de las piezas, los movimientos adecuados y sus consecuencias. Para ello precisa gran capacidad de almacenamiento de información, utilizando estructuras que permitan llegar a conclusiones coherentes.

Así pues, un primer problema es la representación en la máquina del tablero de ajedrez. Claude Shannon sugirió en 1949 que cada casilla del tablero se representará en el computador por una palabra. El contenido de esta palabra será el indicador del tipo de pieza situada sobre la casilla. A cada pieza se le asigna un número entero, haciéndole corresponder a las blancas los números positivos y a las negras los negativos. Se representan los peones por un uno, los caballos por un dos, los alfiles por tres y así sucesivamente. Las casillas vacías se representan mediante un cero. Utilizando este sistema puede averiguarse si una casilla del tablero tiene una pieza o no, y si la tiene cuál es y de qué color.

Para poder detectar fácilmente los bordes del tablero se trabaja con unas dimensiones hipotéticas de 10 por 12 en lugar de las reales de 8 por 8. El tablero real de juego, como se indica en la figura 14, tendrá casillas numeradas entre 22 y 99. Las demás posiciones corresponden a casillas que están fuera del tablero. Estas contendrán un número que no va a corresponder a ninguna pieza, por ejemplo, 99.

Con esta representación pueden determinarse los movimientos permitidos desde cualquier posición, estableciendo una relación matemática entre las distintas casillas. Por ejemplo, un caballo situado sobre cualquier

92	93	94	95	96	97	98	99
82	83	84	85	86	87	88	89
72	73	74	75	76	77	78	79
62	63	64	65	66	67	68	69
52	53	54	55	56	57	58	59
42	43	44	45	46	47	48	49
32	33	34	35	36	37	38	39
22	23	24	25	26	27	28	29

Fig. 14

casilla puede realizar ocho movimientos diferentes. Las casillas a las que podrá moverse se obtienen sumando a la posición en que se encuentra las cantidades: +8, +19, +21, +12, −8, −19, −21 y −12. Si un caballo está en la casilla 99 y se aplica el movimiento correspondiente a sumar +21, pasa a la casilla de la posición 120 que contiene un 99, indicando que está fuera del tablero de juego y, por tanto, el movimiento es incorrecto. Si estuviera en la casilla 59 y se le suma +21, +12, −19 y −8, pasa a las casillas 80, 71, 40 y 51, respectivamente, que están fuera del tablero. De forma similar, los movimientos del rey podrán calcularse a partir de las cantidades: −1, +9, +10, +11, +1, −9, −10 y −11.

Cuando se realiza un movimiento hay que comprobar que éste es válido. Si la posición obtenida contiene el número 99, el movimiento propuesto es incorrecto, la nueva posición está fuera del tablero. Si se juega con blancas y la casilla destino contiene un número positivo, no podrá efectuarse este movimiento por estar ya ocupada por otra pieza blanca. Si contiene un número negativo, la pieza blanca podrá situarse en la casilla, capturando la pieza negra que hay en ella. Si la nueva posición contiene un cero, el movimiento será correcto, pues la casilla está vacía.

Los movimientos de alfil, torre y reina son ligeramente más complicados. Si una parte de un alfil blanco situado sobre la casilla que corresponde a la posición 55 y se quiere estudiar los movimientos correctos sobre

una de las diagonales, deberían tratarse sucesivamente las posiciones 66, 77, 88,... ó 44, 33,... Para cada posición comprobará si está ocupada por una pieza blanca o negra, actuando en consecuencia. Si la casilla está vacía (en la posición hay un cero), puede optarse por moverla a dicha posición o estudiar el movimiento a la posición siguiente en la diagonal. De forma análoga se realizan los movimientos de las torres, y los de la reina son una conjunción de ambos.

Existe otra forma de representar el tablero en el computador. En lugar de asignar una palabra para cada casilla, se asigna dentro de una palabra de 64 bits, a cada casilla un bit. Así puede representarse el tablero con doce palabras. Con una palabra pueden representarse los peones blancos; basta poner los bits a 1 ó 0, según haya o no peón en la casilla correspondiente. Los caballos negros se representan con otra palabra, y así sucesivamente.

Además de las piezas en sí, este método permite representar otro tipo de información. Puede tenerse una palabra que represente todas las casillas en las que hay piezas blancas, otras para las negras, otra para las casillas atacadas por piezas blancas, etc. El programador iniciará todas las relaciones que crea convenientes.

Con esta representación, el computador puede realizar fácilmente operaciones booleanas con Y y O lógicos que son útiles para obtener los resultados deseados.

Supóngase que se estudian, por ejemplo, los movimientos válidos desde una posición determinada de un alfil blanco. Los movimientos de un alfil desde cada una de las 64 casillas están representados en 64 palabras de memoria. Habrá que buscar en memoria la palabra que corresponde a la posición en que se encuentra el alfil, para así tener los posibles movimientos. Seguidamente, habrá que obtener la palabra que representa las casillas ocupadas por piezas blancas. Podrá moverse el alfil a las posiciones en que no haya pieza blanca y sea posible el movimiento. Para ello habrá que negar (cambiar unos por ceros y ceros por unos) la palabra que indica las posiciones de las blancas y aplicar el Y lógico con la palabra que tiene los movimientos posibles.

A la hora de considerar los posibles movimientos realizados, por ejemplo, los de una pieza blanca, habrá que estudiar las réplicas de las negras, así como las contrarréplicas de las blancas, y así sucesivamente, hasta que se determine una posición. Para poder examinar toda la información disponible en el tablero se crea una función de evaluación. Esta estudiará la conveniencia de una posición en un momento determinado.

La función de evaluación va a sopesar una serie de factores que, dependiendo de su importancia, influirán en mayor o menor grado en la determinación del movimiento.

Estos factores pueden ser entre otros: las piezas en el tablero, la estructura de los peones, el control del centro del tablero, el ataque a posiciones

adyacentes al rey contrario, el cierre de piezas, la posibilidad de intercambio de piezas, la seguridad del rey propio, etc. Cada vez que se evalúa una posición habrá que calcular el valor de esta función. Eso es muy frecuente en un programa de ajedrez, por lo que la función de evaluación debe implementarse con instrucciones sencillas y no es conveniente incluir términos para todos los factores.

Una buena función de evaluación es aquélla que considera los aspectos críticos de la posición y los trata de una forma adecuada. Al incluir un factor en la función de evaluación habrá que considerar si la eficacia introducida por dicho factor compensa el aumento producido en el tiempo de ejecución. La función de evaluación depende del método utilizado para determinar el movimiento.

Es difícil determinar cuándo un nodo puede considerarse como terminal y, por tanto, susceptible de aplicar sobre él la función de evaluación. En general, se considera que un nodo es terminal cuando a partir de él no existen más capturas.

Para determinar la siguiente posición habrá que generar todos los movimientos posibles, todas sus réplicas y contrarréplicas hasta llegar a una posición del tablero que considere el movimiento como idóneo.

No es posible realizar una búsqueda completa en profundidad porque se tiene un elevado número de combinaciones (del orden de 10^{120}); por tanto, habrá que fijar un nivel máximo de profundidad.

Otra forma de solucionar este problema es examinar sólo un subconjunto de lo movimientos posibles en cada nodo. Se utiliza el método Neumann y Morgenstern, conocido como Minimax. Si quiere reducirse aún más el árbol de búsqueda, se emplea el método de poda alfa-beta.

Los programas de ajedrez actuales dan gran importancia a la determinación del primer movimiento que va a estudiarse en cada nodo del árbol. Para ello, se utilizan métodos heurísticos que proporcionan información sobre el tipo de movimientos que se le aplican en una situación particular acelerando la poda. Se toma el criterio de elegir los movimientos en que se realice la captura de una pieza. Esto provoca un aumento en la poda, ya que se reduce el número de piezas y, por tanto, el número de réplicas del contrario.

Otra heurística consistiría en contestar a un movimiento con otro ya estudiado en otra parte del árbol, pues a lo largo de la partida se repiten muchos movimientos. En este caso, se utilizan analogías. La estrategia consiste en examinar con detalle las réplicas y determinar qué factores son necesarios para que estas réplicas sigan siendo válidas.

Todos los métodos vistos hasta ahora no son eficientes a la hora de seleccionar el tipo de salida. La efectividad de una salida no puede comprobarse hasta que la partida esté muy avanzada. Debido a que el número de réplicas y contrarréplicas que se estudian para cada movimiento tienen una profundidad determinada, a ese nivel todavía no puede comprobarse

la bondad de una salida, por lo que se utilizan librerías de salidas ya almacenadas en el computador.

De igual forma, el fin de juego es un caso especial y requiere información adicional. La estrategia ganadora conlleva una secuencia de veinte o más movimientos. Una búsqueda completa para esta profundidad es totalmente imposible. Por este motivo debe abandonarse el procedimiento Minimax con poda alfa-beta al final del juego y adoptar una estrategia alternativa. La solución es disponer de una librería de instrucciones para los finales de juego.

ANIMALES

En este juego aparecen dos protagonistas, el jugador (usuario) y el computador. La mecánica del juego consiste en pensar un animal y el computador irá realizando una serie de preguntas relevantes que le permitan descubrir de qué animal se trata. El jugador deberá responder a estas preguntas mediante una afirmación (SI) o (NO).

El desarrollo del juego requiere la existencia de una Base de Datos. En ella estarán contenidas todas las preguntas que pueden hacerse al usuario, así como una serie de animales que serán las posibles respuestas.

La Base de Datos estará organizada de tal forma que dependiendo que la respuesta a un pregunta sea afirmativa o negativa se bifurcará a una zona determinada de la Base. Las preguntas contenidas en esta zona estarán en relación con la respuesta a la pregunta anterior. En este juego pueden introducirse técnicas de aprendizaje. Entre las distintas técnicas de aprendizaje disponibles en IA puede utilizarse la más sencilla, el aprendizaje por instrucción. Este tipo de aprendizaje se caracteriza por que el conocimiento se adquiere a través de un profesor u otras fuentes como pueden ser los libros de texto.

Aplicado a este caso, el aprendizaje se realizará cuando el computador no tenga en su Base de Datos el animal que el jugador ha pensado. El programa le preguntará al jugador de qué animal se trata, introduciéndolo posteriormente en su Base de Datos. También pedirá al jugador que enuncie la pregunta que debería hacerse para definir las características que diferencian a este animal.

Seguidamente se proponen una Base de Datos que puede ampliar el lector y un programa que realiza las preguntas.

Un base de animales sencilla puede estar organizada de la siguiente forma:

```
( ( ES VERTEBRADO )
   ( ( VUELA )
      ( GUSANO )
```

```
    ( MOSQUITO ) )
  ( ( ES UN ANIMAL DE SANGRE CALIENTE )
    ( ( TIENE BRANQUIAS Y SIEMPRE VIVE EN EL AGUA )
      ( ( TIENE BRANQUIAS QUE SE TRANSFORMAN EN PULMON )
        ( ( TIENE PATAS )
          ( SERPIENTE )
          ( COCODRILO ) )
      ( ATUN ) )
    ( ( SE CRIA CON LECHE )
      ( ( VUELA )
        ( POLLO )
        ( PETIRROJO ) )
      ( ( VIVE EN EL AGUA )
        ( ( ES UN ANIMAL DOMESTICO )
          ( TIGRE )
          ( PERRO ) )
        ( DELFIN ) ) ) ) ) )
```

La base tiene una estructura en forma de lista y dependiendo de que la respuesta a la pregunta sea afirmativa o negativa pasará a estudiarse el CADR o el CDDR, respectivamente, de la lista para hacer una nueva pregunta. Como puede verse en la función:

```
( DEFUN  PREGUNTAR ( L )
  ( COND
    ( ( NOT ( ATOM L ) )
      ( PRINT ( CAR L ) ( QUOTE ? ) )
      ( SETQ RESP ( READ ) )
      ( COND
        ( ( EQUAL RESP 'N )
          ( PREGUNTAR ( CADR L ) ) )
        ( ( EQUAL RESP 'S )
          ( PREGUNTAR ( CDDR L ) ) )
        ( T ( PRINT '( LA RESPUESTA NO HA SIDO CORRECTA ) ) ) ) )
    ( T ( COND ( ( EQUAL RESP 'N )
              (PRINT '(ME RINDO NO SE DE QUE ANIMAL SE
              TRATA) ) ) ) ) ) )
```

Se sugiere al lector que mejore la función anterior introduciendo el aprendizaje.



CUATRO EN RAYA

Las cuatro en raya se juega sobre un tablero de siete por siete, donde se colocan las fichas de dos colores, correspondientes a cada jugador. Las fichas se dejan caer por cada columna de forma alternativa. El objetivo es

conseguir alinear cuatro fichas del mismo color, ya sea vertical, horizontal o diagonal.

En un juego de estrategia como éste, siempre se trata de saber si se dispone de ventaja sobre el contrincante y cómo se puede mejorar ésta. El programa debe poder evaluar las consecuencias de un posible movimiento. Esto, que para una persona puede resultar complicado, es fácil para un computador. Siempre y cuando se encuentre una estructura adecuada de representación y un método de evaluación de cada movimiento lo más acertado posible.

En este juego se puede ver la imposibilidad de ganar en pocas jugadas; se debe tratar de ir formando alineamientos de dos fichas, si es posible de tres. Cuantas más alineaciones se tengan, más probabilidad de ganar. Es evidente que los alineamientos de tres fichas tienen más valor que los de dos; además, hay que impedir que el contrario efectúe dichos alineamientos, o por lo menos estorbarlos.

Este juego se presta a ser tratado con el procedimiento Minimax, es decir, evaluar una jugada y, a continuación, todas las posibles respuestas del adversario, y así sucesivamente hasta encontrar la que más posibilidad de éxito presente. Para evaluar la eficacia de cada movimiento se debe utilizar una función adecuada. En ese caso se utilizará la siguiente función:

$$W(M) = 4^{*} N_2 + 9^{*} S_3$$

donde N_2 es el número de alineamientos de dos fichas que aparecen al ejecutar el movimiento M y N_3 el número de alineamientos de tres fichas. Estos dos valores están multiplicados por un factor que corresponde a la importancia de cada tipo de alineamiento. Además, si se consigue un alineamiento de cuatro fichas la función W toma el valor máximo o el mínimo, según sea un alineamiento del computador o del contrincante.

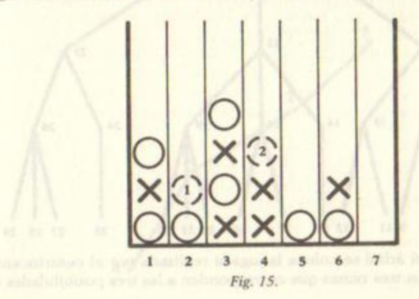


Fig. 15.

Esta función se utiliza de la siguiente manera: en cada posición que ensaya el ordenador busca todas la posibles cadenas de longitud dos y todas las de longitud tres que se realizan al colocar la ficha. Si el juego está en la posición de la figura 15 el computador juega con los círculos y el contrincante con cruces, la evaluación correspondiente a poner un círculo en la columna cuarta será:

$$W_1 = 4 * 1 + 9 * 1 = 13$$

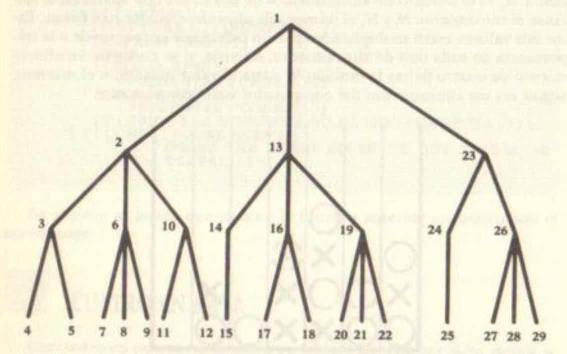
Si se pone el círculo en la columna 2, la función de evaluación sería:

$$W_2 = 4 * 4 + 9 * 0 = 16$$

ya que en este caso daría lugar a cuatro alineamientos de dos fichas.

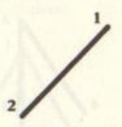
En principio, según la evaluación, sería más conveniente realizar la 2.ª jugada. Pero la primera podría llevar a conseguir un alineamiento de cuatro en el siguiente movimiento. Para poder dirimir en este tipo de casos es para lo que se hace la simulación de las respuestas del adversario, y de las de la máquina al adversario, hasta un nivel determinado. Para la ejecución del algoritmo serán necesarias unas variables S_i que son los tanteos que potencialmente obtiene cada nodo para el camino estudiado. S_0 y S_2 se inicializan a $-\infty$, mientras que S_1 se inicializa a $+\infty$ cada vez que se ha encontrado un nodo en el nivel correspondiente.

Como mejor se comprenderá el mecanismo de la elección de movimiento es desarrollando un ejemplo:

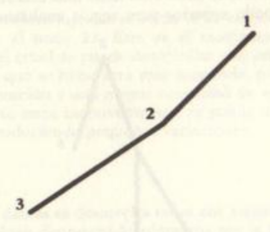


En la raíz del árbol se coloca la jugada realizada por el contrincante. De esta raíz salen tres ramas que corresponden a las tres posibilidades de

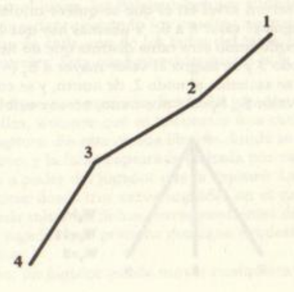
respuesta que tiene la máquina, y de cada una de ellas salen las posibles respuestas del contrincante. La forma de recorrer el árbol es la siguiente:



Se escoge la primera respuesta y se analiza:

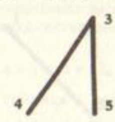


De este nivel toma la primera respuesta del adversario y de aquí, la primera respuesta de la máquina:



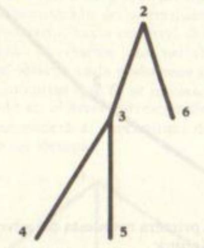
Llegado a este punto, el computador evalúa la función para el movimiento efectuado; supóngase que W_4 es mayor que $-\infty$ y S_2, toma el valor 4.

A continuación se sube al nodo 3 y se explora el nodo 5



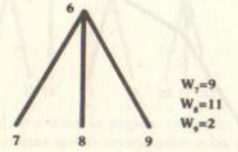
si, por ejemplo, W_5 es 11, este valor es mayor que S_2 (= 4); por tanto, el nuevo valor de S_2 es 11.

Llegado a este punto, el nodo 3 no tiene más descendientes y, por tanto, se asciende al nodo superior para seguir analizando otras variantes:

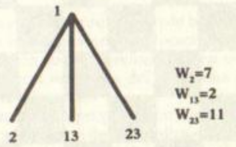


Antes de pasar al nodo 6, hay que comparar el valor de S_2 con el de S_1. S_1 tiene valor ∞ por ser un nivel en el que se quiere minimizar la función. Como $S_2 < S_1$ se asigna el valor 8 a S_1. Y además hay que hacer $S_2 = \infty$, ya que ahora se está explorando otra rama distinta que no tiene nada que ver con la rama del nodo 3 y se asigna el valor mayor a S_2 (=11).

Con este valor, se asciende al nodo 2, de nuevo, y se compara el nuevo valor de S_2 con el valor S_1, $S_2 < S_1$; por tanto, no se cambia.



Cuando se ha terminado de explorar todas las ramas del nodo 2, se tiene almacenado en S_1 el valor mínimo de los tres valores de sus nodos descendientes. A continuación, se repite la operación con el resto de las ramas que parten de la raíz y se obtiene:



De estas tres posibles jugadas se toma la de mayor valor, que en este caso supondría ir al nodo 23. Este es el movimiento que se ejecuta.

Por supuesto, el árbol se puede desarrollar con más profundidad, con lo que la decisión que se tome será más aceptable, pero esto supone mayor tiempo de operación y una mayor necesidad de espacio en memoria. Para evitar en parte estos inconvenientes, se puede utilizar el método de poda alfa-beta introduciendo pequeñas variaciones.



DAMAS

El juego de las damas se desarrolla entre dos jugadores sobre un tablero de ajedrez, también denominado «damero» por la práctica habitual de este juego. Se utilizan 24 fichas redondas y planas distribuidas en número igual entre los colores blanco y negro, de forma que se reparten, quedando cada jugador en posesión de 12 fichas del mismo color.

Para comenzar el juego, las fichas deben ser situadas sobre el tablero delante de cada jugador, ocupando los cuadros negros. De esta forma se ocupan tres líneas de cada uno de los lados del tablero, quedando vacías las dos líneas centrales. Esta configuración inicial es la que se muestra en la figura 16.

La finalidad del juego es tomar las fichas del jugador contrario, saltando por encima de ellas, siempre que se encuentre una casilla vacía detrás de la ficha que se captura. En esta casilla libre es donde se situará la ficha del jugador que mueve, y la ficha capturada (saltada por encima) se retira del tablero, pasando a poder del jugador que la capturó. La captura de una ficha puede repetirse dos o tres veces seguidas en el turno de un jugador siempre que pueda saltar las fichas correspondientes del adversario. Gana la partida aquel jugador que primero consigue apoderarse de todas las fichas del oponente.

En cada turno, un jugador puede mover cualquiera de sus fichas, pero

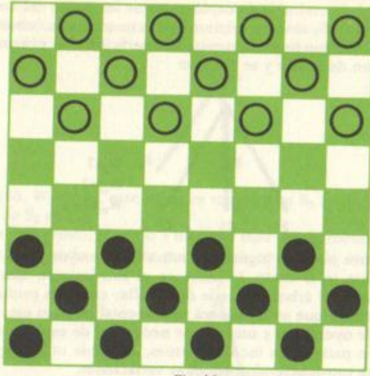


Fig. 16.

sólo una de ellas. Un movimiento consiste en desplazar la ficha un cuadro hacia delante, pero en dirección diagonal, es decir, a otro cuadro negro adyacente situado hacia delante. Nunca puede moverse una ficha en línea recta ni hacia atrás, salvo en el caso de que la ficha que se mueve sea una dama, en cuyo caso puede mover hacia atrás diagonalmente.

Un jugador consigue una «dama» cuando ha llegado con una de sus fichas a la primera línea del campo adversario, es decir, al borde opuesto del tablero. Para reconocer a una dama se le coloca otra ficha encima. A partir de ese momento la dama es la única ficha que tiene derecho a desplazarse a lo largo del tablero, todas las casillas que pueda hasta llegar a un borde o a una ficha propia, tomando todas las fichas del bando contrario que encuentre en su camino. La potencia de una dama es superior a la de una ficha normal, por lo que hay que intentar hacer dama lo más rápidamente posible.

Una jugada interesante es aquélla en la que uno de los jugadores realiza un movimiento de forma que deliberadamente deja una de sus fichas en peligro, ya que su adversario puede capturarla en un movimiento. Si éste omite comer la ficha en su turno, se dice que ésta se encuentra en «soufle» y se retira del juego como si el jugador realmente la hubiera comido. Esta actitud, sin embargo, no es obligatoria y sólo se hace uso de ella si la situación resultante es ventajosa.

Respecto a la programación de este juego existen diversos programas que ofrecen comportamientos diversos. El más famoso y cuidado de ellos

es el realizado por Arthur Samuel en la Universidad de Illinois. Comenzó sus primeros trabajos en el estudio del juego de las damas en 1947, año en el que apareció la primera versión de su programa. Sus investigaciones continuaron realizando versiones mejoradas del programa hasta la última, que está fechada en 1967.

El programa se basa en un procedimiento de búsqueda alfa-beta que utiliza purga convergente hacia delante y exploración del árbol en amplitud para realizar la ordenación de los movimientos en orden decreciente de plausibilidad. Las diferentes versiones del programa incorporan diversas heurísticas para guiar la búsqueda.

La gran importancia de este programa consiste en su utilización para investigar en el intrincado campo del aprendizaje por parte de los computadores. En términos generales, puede decirse que el programa incorpora una combinación de las técnicas denominadas «aprendizaje por generalización», partiendo de ejemplos para concluir reglas generales de operación y «aprendizaje memorístico», es decir, almacenamiento ordenado de movimientos o acciones que son de importancia por su trascendencia en el desarrollo del juego o por su frecuencia de aparición.

En lo que se refiere a aprendizaje por generalización, el programa consigue mejorar la función de evaluación estática del procedimiento alfa-beta. Para ello utiliza procedimientos prácticos que llevan al computador a aprender buenas funciones de evaluación por medio de análisis estadísticos sobre las jugadas que aparecen en los libros escritos por expertos. El trabajo habitualmente debe realizarse partiendo de la información directa obtenida de los expertos en la materia, pero si se carece de la presencia de un experto no queda más remedio que utilizar la información que ofrecen los libros especializados.

Estos libros especializados suelen mostrar una serie de movimientos del juego en determinadas situaciones que conducen siempre, o casi siempre, a la victoria. La función de evaluación, entonces, ha de diseñarse de forma que seleccione los movimientos recomendados por los expertos con la mayor frecuencia posible. En sus primeras investigaciones, Samuel restringió las funciones posibles exclusivamente a funciones lineales. Posteriores versiones del programa hacen uso de lo que denomina «tabla de signaturas», en la que se registran determinados parámetros que afectan a cada jugada, así como la propia jugada. Para trabajar con esta tabla no es necesario que la función de evaluación sea lineal, ya que ha de tener en cuenta la posible interacción entre distintos factores. Existe una enorme dificultad práctica en encontrar la función de evaluación óptima, es decir, la función que elige siempre el movimiento adecuado (el recomendado por los libros). Sin embargo, Samuel consiguió programar procedimientos que construían funciones de evaluación que elegían el movimiento correcto en un 38 por 100 de las ocasiones, aproximadamente. La última versión del programa consigue porcentajes aún mejores.

En lo que se refiere al aprendizaje por memorización, el programa permite «aprender» movimientos recomendados por los libros, al igual que posiciones que ya han sido evaluadas por el programa y que suceden con frecuencia en las partidas reales. El programador puede forzar al ordenador a elegir en alguna situación concreta uno de los movimientos que tiene almacenados en la memoria, si existe alguno aplicable. También puede dejarse al programa elegir por sí mismo el momento adecuado en el que debe realizar uno de los movimientos aprendidos. Cada vez que el programa es forzado a una jugada de las de su memoria, recuerda que el movimiento no fue realizado como consecuencia de la búsqueda, sino extraído directamente de la memoria.

El mercado informático posee juegos de damas comercializados de muy diversas prestaciones; por supuesto, todos ellos omiten toda la parte de aprendizaje que se ha expuesto, ya que sus fines son meramente comerciales y no de investigación.

Variantes del juego

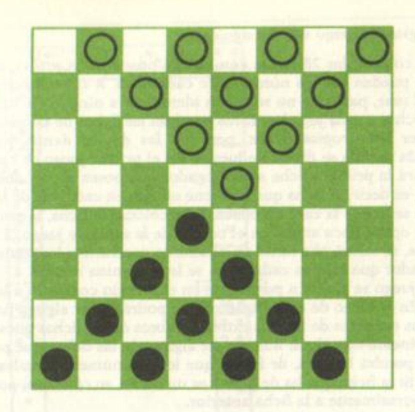
El juego de las damas está ampliamente extendido y quizá por ello, existen un gran número de variantes sobre el juego original. La mayoría de las variaciones son bastante poco interesantes, tanto desde el punto de vista computacional como desde el punto de vista lúdico, por lo que no han tenido gran aceptación por parte de la comunidad informática. Una de las variantes más famosa e interesante son las damas triangulares, cuyas reglas se explican a continuación.

El juego de las damas triangulares utiliza también el tablero de ajedrez, pero sólo son necesarias 20 fichas, 10 para cada uno de los dos jugadores que intervienen. El nombre del juego proviene de la colocación de las fichas sobre el tablero en el comienzo del juego, en forma de triángulo, como muestra la figura de la página siguiente.

Las fichas se mueven de la misma forma que en el juego de las damas, pero la ficha del contrario que es saltada no resulta afectada por ello, es decir, no se atrapan las fichas del oponente. La finalidad del juego es la colocación de todas las fichas de un jugador en las posiciones que ocupaban inicialmente las del oponente, ganando el jugador que lo consiga en primer lugar.

En este caso la estrategia del juego no consistirá en perseguir implacablemente las fichas del contrario, sino que debe buscarse la situación de las fichas propias sobre el tablero de forma que obstaculicen el paso del adversario a la vez que se van acercando a las posiciones buscadas.

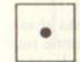
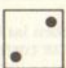
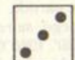
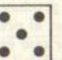
Como puede observarse, la realización de un programa sobre este juego no diferiría demasiado de la formulación del programa de las damas. Básicamente se utilizaría cualquier implementación del procedimiento de poda alfa-beta, que resulta el más adecuado para este tipo de juegos.



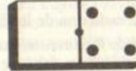


EL DOMINO

Pese a que existen muchas variantes del juego, este capítulo sólo se va a centrar en la variante clásica que se practica entre dos parejas de jugadores. El componente básico del dominó son las 28 fichas de dos caras con las que se juega. Una de las caras es opaca, de forma que no se puede ver lo que contiene la otra cara. Esta última es la que posee la información de la ficha, y está separada por la mitad por una línea divisoria. Cada una de estas partes puede estar en blanco o poseer un número de puntos contenido entre uno y seis. Estos puntos negros vienen expresados de la siguiente forma:

Por tanto, una ficha como el blanco-cuatro vendrá expresada de la siguiente forma:



Las reglas del juego son las siguientes:

1. Se colocan las 28 fichas con la cara opaca hacia arriba, de forma que no se puedan ver los números de cada ficha. A continuación, se revuelven al azar, para que no se pueda identificar a ninguna ficha, y se reparten 7 fichas a cada jugador. Estos pondrán las fichas de tal manera que puedan ver sus propias fichas, pero no las de los demás jugadores.

2. Cada partida se divide en juegos. En el primer juego de cada partida, colocará la primera ficha aquel jugador que posea el seis doble entre sus fichas; es decir, la ficha que contiene un seis en cada una de las partes en las que se divide la cara no opaca. Para colocar la ficha, la pondrá con la cara no opaca boca arriba en el centro de la mesa de juego. En los demás juegos, se saldrá por turno en el sentido contrario a las agujas del reloj. Al jugador que sale en cada juego se le denomina mano.

3. El juego se practica por turno, en el sentido contrario a las agujas del reloj. En el turno de cada jugador, éste podrá poner alguna ficha si alguno de los números de los dos extremos libres de las fichas puestas en el tablero coincide con algún número de alguna de las fichas que posee. En este caso, pondrá la ficha, de forma que los dos números iguales queden adosados. Si la ficha que ha de poner es un doble, se colocará, por convenio, transversalmente a la ficha anterior.

En la figura de la página siguiente se puede observar la situación de una partida al final de la misma. Puede comprobarse que el seis doble no se ha puesto, por lo que se deduce que no es el primer juego de una partida. Si algún jugador no puede poner ninguna de sus fichas, se dice que pasa, y le tocará jugar al siguiente a él.

4. El juego puede terminar de dos formas:

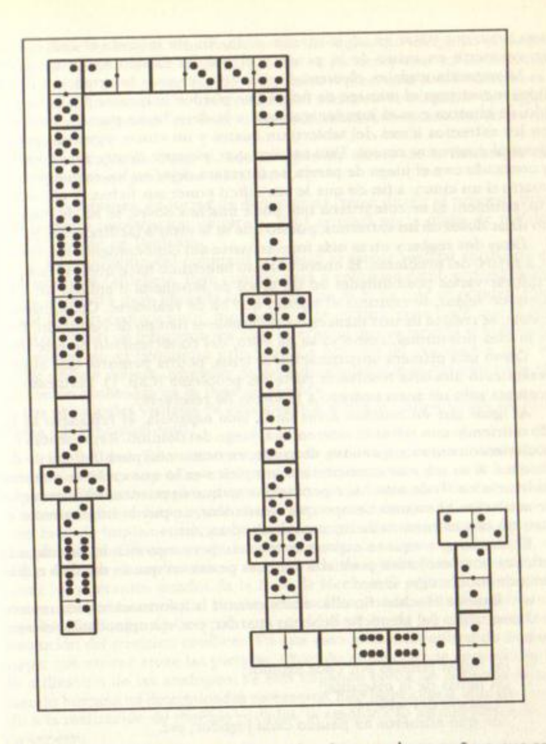
a) Todos los jugadores pasan, con lo cual se dice que la partida se ha cerrado. Ganará la pareja que menos puntos posea entre sus fichas.
b) Algún jugador termina de poner todas sus fichas, con lo cual ganará la pareja formada por él y su compañero.

Normalmente, se establece un convenio de puntuación. Así en cada juego se van acumulando puntos hasta que se llegue a un tope acordado. Cuando esto se produzca, se finalizará la partida.

Debido a que, al principio de cada juego, se reciben las fichas al azar, se puede pensar que este juego es de azar. Esto no se corresponde totalmente con la realidad, puesto que se ha de saber jugar y utilizar las técnicas que existen. Las reglas empíricas primordiales sobre las que actúan casi todos los buenos jugadores son dos:

— jugar *con* el compañero;
— recordar quién ha puesto cada una de las fichas y en qué momento.

La primera regla se comprende fácilmente, pues el objetivo es derrotar a una pareja y para ello es necesaria la colaboración de los dos miembros



del equipo. En principio, se ha de ayudar siempre al compañero que sea el mano o, en su defecto, que esté a la derecha del mano. Se entiende por ayudar el colocar fichas en cuyo extremo libre queden números sobre los que el compañero haya demostrado estar fuerte. Sin embargo, existen situaciones en las que se intercambian los papeles de mano dentro de la pareja. Esto sucede, por ejemplo, cuando el que actúa como mano en la pa-

reja tiene que pasar en un momento dado. En ese instante el compañero se convierte en mano de la pareja y el que ha pasado ha de ayudarle.

La segunda regla es elemental, pues si se conoce información de tal tipo, se restringe el número de fichas que pueden colocarse. Si, por ejemplo, se observa que el jugador que está a la derecha ha pasado, teniendo en los extremos libres del tablero un cuatro y un cinco, significa que no tiene ni cuatros ni cincos. Esto implica que, siempre dentro de lo posible y contando con el juego de pareja, se intentará dejar en los extremos o un cuatro o un cinco, a fin de que le sea difícil poner sus fichas. Y al contrario, también. Si se comprueba que pone muchos doses, se ha de intentar no dejar doses en los extremos, puesto que se le estaría facilitando el juego.

Estas dos reglas y otras más forman parte del conocimiento heurístico o, *a priori*, del problema. El conocimiento heurístico hace que a la hora de explorar varias posibilidades en un árbol de búsqueda o aplicar un conjunto de reglas, se restrinja el trabajo que ha de realizarse. Consiguientemente, se reduce de una manera considerable el tiempo de ejecución. Esta es la idea primordial, como ya se ha visto, del conocimiento de este tipo.

Como una primera aproximación al tema, podría pensarse que el procesamiento alfa-beta resolvería parte del problema (Cap. 1). Pero esto representa sólo un acercamiento a la forma de resolverlo.

Al igual que en muchas áreas de la vida española, el refranero se ha ido nutriendo con refranes referentes al juego del dominó. Representan conocimiento empírico que sirve de ayuda, en ocasiones, para la toma de decisiones. A su vez este conocimiento empírico es lo que en IA se denomina heurística. Todo esto hace pensar que ya que se pueden obtener reglas de actuación al mismo tiempo que heurísticas, se puede implementar el dominó con estructura de Sistema de Producción.

El diseño que aquí se expone es teórico, pero está siendo llevado a la práctica con resultados positivos. Las tres partes en que se dividirá el Sistema de Producción serán:

a) Base de Hechos. En ella se almacenará la información concerniente al desarrollo del juego. Se deberán guardar, por ejemplo, datos referentes a:

— las fichas puestas por cada jugador;
— las fichas de la máquina;
— situación del tablero;
— con qué números ha pasado cada jugador, etc.

En este caso, el estado inicial es el principio de la partida. El estado final no está definido en la Base de Hechos, y aparecerá en la Base de Reglas. Sin embargo, se sabe hacia qué estado se tiende: lograr colocar todas las fichas o ayudarle al compañero a que lo haga. Como una segunda meta puede considerarse el cerrar cuando se crea que se tienen menos puntos entre la máquina y su compañero, que los contrarios.

b) Base de Reglas. Contendrá todas las reglas que dirigirán el juego. Al mismo tiempo, delimitarán qué fichas se pueden poner en un instante determinado. Se dispone también de metarreglas, que son los refranes. Como ya se ha explicado, las metarreglas reducen el conjunto de reglas que han de aplicarse, en base a determinadas informaciones sobre el problema.

c) Estrategia de Control. Los dos puntos claves de la Estrategia de Control para este problema van a ser:

— Los refranes, en forma de metarreglas, van a dirigir el proceso, proporcionando información empírica. Esto ayudará en la etapa de restricción en la fase de Decisión.

— La resolución del conjunto conflicto se puede realizar mediante el procedimiento alfa-beta. Para cuando vaya a aplicarse dicho procedimiento, ya se tendrá muy limitada la elección de la ficha que se pondrá. Ello ayudará al procedimiento y, por tanto, al programa, pues hace que el número de ramas de búsqueda sea muy inferior al que se tendría si fuese este procedimiento el programa principal. La función de evaluación que se puede utilizar deberá depender de probabilidades que se basarán en la información contenida en la Base de Hechos.

Las principales ventajas de esta solución sobre la utilización de un programa convencional son:

— flexibilidad. Este sistema trata cada partida de forma diferente, al mismo tiempo que se adapta al transcurso de la partida. En cada juego se disparan diferentes reglas, dependiendo de la situación de la partida.

— aprendizaje. Se le puede añadir un proceso de aprendizaje. Esto resulta fácil de implementar, ya que se puede guardar una historia del proceso del juego. Esto quiere decir que se puede almacenar, por ejemplo, el orden de las reglas aplicadas en cada partida. Al mismo tiempo, se pueden salvar los diferentes estados de la Base de Hechos. Si jugando una partida se encuentra una situación similar a una almacenada, evitará mucho trabajo a la Estrategia de Control, tanto en la elección de las reglas como en resolución del conjunto conflicto. En este caso se estarán utilizando las analogías que existen entre las partidas. Uniendo el proceso de aprendizaje al de utilización de las analogías, se está llegando cerca de la forma de actuación humana en determinados momentos. Ello quiere decir, que con arreglo a la realización de muchas partidas, la máquina se irá convirtiendo en un experto.



ELISA

Se trata de simular una conversación entre un psiquiatra (la máquina) y un supuesto paciente (el jugador). Esta conversación se llevará a cabo

en base a modelos de frases preestablecidos. El programa debe encontrar la semejanza entre las frases modelos y la frase que el paciente ha introducido por el terminal.

El problema que hay que resolver es determinar las semejanzas entre el patrón y la frase en estudio. Habrá que comparar expresiones para ver si son similares. Aunque el lenguaje LISP no tenga preestablecidos patrones, pueden implementarse fácilmente. Se considerarán el patrón y toda o parte de la frase que se compara como listas; serán, pues, una lista de átomos.

Un patrón puede contener símbolos especiales que no están permitidos en las frases. Cuando se compara un patrón, que no contiene ninguno de estos símbolos especiales, con una frase, coincidirán sólo si son exactamente iguales, es decir, los átomos (palabras) en las mismas posiciones deben ser iguales. Se puede utilizar el símbolo especial ? como comodín para casar cualquier átomo y el símbolo + para casar uno o más átomos.

Por ejemplo, los patrones siguientes casan con las frases a su derecha:

```
( COLOR ? ROJO )     ( COLOR MANZANA ROJO)
( ESTO + LISTA )     ( ESTO ES UNA LISTA)
```

Para comparar dos expresiones habrá que hacerlo átomo a átomo; para ello se define la función COMPARAR. Esta función tendrá una estructura recursiva; comparará el primer elemento de las dos listas y, en caso de ser iguales, se llama recursivamente para comparar el resto de los elementos.

En la primera cláusula de la condición,comprueba si se ha llegado al final de la lista y, por tanto, se termina la recursión y la segunda comprueba si una lista es más larga que la otra. Habrá que considerar también los comodines + y ? En la cláusula de la condición en que se comprueba si aparece el comodín + habrá que hacer llamadas recursivas a COMPARAR para ver si se da uno de los casos: hay que casar a + con un átomo o hay que hacerlo con dos o más átomos, en cuyo caso se seguirán haciendo llamadas recursivas.

```
( DEFUN COMPARAR ( PATRON FRASE )
     ( COND    (( AND (NULL PATRON ) ( NULL FRASE ))  T )
               (( OR  (NULL PATRON ) ( NULL FRASE ))  NIL )
               (( OR ( EQUAL ( CAR PATRON ) '? )
                      ( EQUAL ( CAR PATRON ) ( CAR FRASE )))
               ( COMPARAR ( CDR PATRON )  ( CDR FRASE )))
               (( EQUAL ( CAR PATRON ) '+ )
               ( OR ( COMPARAR ( CDR PATRON ) ( CDR FRASE ))
                    ( COMPARAR PATRON ( CDR FRASE ))))))
```

Este programa puede mejorarse introduciendo variables de patrón que van precedidas por > o +, que realizan una función semejante a los comodines ? y + vistos anteriormente. Cuando una variable de patrón va precedida por > al compararse le asocia a la variable el átomo situado en esa posición. Si va precedido por +, asocia uno o más átomos.

Por ejemplo,

```
(EXP ( < A) ( > B)) → (EXP 2 3)
    ((A 2) (B 3))
((+ L) MAMA (+ R)) → (DESDE QUE MAMA HABLO)
    ((L (DESDE QUE)) (R (HABLO)))
```

Como puede verse, las variables por > se asocian a átomos y las precedidas por + se asocian a listas.

Se introduce un tercer argumento en la función COMPARAR, que va a devolver una lista con la variable de patrón y sus valores asociados.

Para generalizar la función COMPARAR se utilizan dos funciones en la cláusula del COND, denominadas INDICADOR-PATRON y VARIABLE-PATRON.

Si el indicador de patrón es >, devolverá como argumento una lista formada por el par formado por la variable de patrón y su valor asociado. Si el indicador de patrón es +, devolvería en dicha lista la variable de patrón y la lista de elementos. Para realizar esto, se usan ASOCIAR-ATOMO y ASOCIAR-LISTA.

Se pasa a exponer dichas funciones:

```
( DEFUN INDICADOR-PATRON ( LISTA )
     ( CAR LISTA ) )

( DEFUN VARIABLE-PATRON ( LISTA )
     ( CADR LISTA ) )

( DEFUN ASOCIAR-ATOMO ( VARIABLE ELEMENTO LISTA )
     ( APPEND LISTA ( LIST ( LIST VARIABLE ELEMENTO ) ) ) )

( DEFUN ASOCIAR-LISTA ( VARIABLE ELEMENTO LISTA )
(COND ((NULL LISTA)( LIST ( LIST VARIABLE (LIST ELEMENTO))))
     ( ( EQUAL VARIABLE ( CAAR LISTA ) )
     (CONS
        (LIST VARIABLE (APPEND (CADAR LISTA) (LIST ELEMENTO)))
        ( CDR LISTA) ) )
     ( T ( CONS ( CAR LISTA )
              (ASOCIAR-LISTA VARIABLE ELEMENTO (CDR LISTA)))))
```

A veces hay que tener en cuenta que dentro de un patrón dos variables han de tomar el mismo valor. Por ejemplo:

ESA CASA ES GRANDE. ME GUSTA ESA CASA.
ESA A ES GRANDE. ME GUSTA ESA A.

Esto se indica con la variable precedida por <. Para ello se utiliza la función SACAR.

```
( DEFUN  SACAR ( VARIABLE LISTA )
        ( CADR ( ASSOC VARIABLE LISTA ) ) )
```

La función COMPARAR quedaría como sigue:

```
( DEFUN COMPARAR ( PATRON FRASE ASIGNACION )
        ( COND
            ( ( AND (NULL PATRON ) ( NULL FRASE ) )
              ( COND ( ( NULL ASIGNACION )  T )
                     ( T  ASIGNACION ) ) )
            ( ( OR  (NULL PATRON ) ( NULL FRASE ) )   NIL )
            ( ( OR ( EQUAL ( CAR PATRON )  '? )
                  ( EQUAL ( CAR PATRON ) ( CAR FRASE ) ) )
              (COMPARAR (CDR PATRON) (CDR FRASE) ASIGNACION) )
            ( ( EQUAL ( CAR PATRON )  '+ )
              (OR (COMPARAR ( CDR PATRON ) ( CDR FRASE ) ASIGNACION)
                  ( COMPARAR PATRON ( CDR FRASE ) ASIGNACION) ) )
            ( ( ATOM ( CAR PATRON ) ) NIL )
            ( ( EQUAL (INDICADOR-PATRON ( CAR PATRON ) ) ' ) )
              (COMPARAR ( CDR PATRON ) ( CDR FRASE )
                  (ASOCIAR-ATOMO (VARIABLE-PATRON (CAR PATRON))
                                 (CAR FRASE)
                                 ASIGNACION ) ) )
            ( ( EQUAL (INDICADOR-PATRON ( CAR PATRON ) ) ' ( )
              (COMPARAR
                (CONS
                  (SACAR (VARIABLE-PATRON (CAR PATRON)) ASIGNACION)
                         (CDR PATRON))
                  D
                  ASIGNACION) )
            ( ( EQUAL (INDICADOR-PATRON ( CAR PATRON ) ) ' + )
              (LET (ASIGNUEVA (ASOCIAR-LISTA
                               (VARIABLE-PATRON (CAR PATRON))
                               (CAR FRASE)
                               ASIGNACION) ) )
                  (OR (COMPARAR (CDR PATRON) (CDR FRASE) ASIGNUEVA)
                      (COMPARAR PATRON (CDR FRASE) ASIGNUEVA) ) ) )))
```

En base a lo anterior, puede escribirse una función que simule una conversación coloquial, como puede ser:

(DOCTOR)
(HABLE)
(ESTOY PREOCUPADO POR MI SALUD)
(CUANTO TIEMPO HACE QUE ESTA PREOCUPADO)
(DESDE QUE MI MADRE HABLO)
(HABLEME DE SU FAMILIA)
(ACTUALMENTE TODOS TRABAJAN CON ORDENADORES)

(TIENE MIEDO A LAS MAQUINAS)
(NO)
(NO ME OCULTE NADA, POR FAVOR)
(VAYASE AL INFIERNO)
(POR FAVOR, NO UTILICE ESE VOCABULARIO)
(ME ESTA FRUSTRANDO)
(HABLE DE SU MADRE)
(AHORA NO – DIGAME CUAL ES MI PROBLEMA)
(LO SIENTO LA SESION HA FINALIZADO)
ADIOS

La función DOCTOR será un bucle controlado por un COND que realiza la comprobación de palabras en las frases dando las respuestas que se sugieren en el diálogo anterior. Se sugiere al lector que realice dicho programa.



KALAH

El kalah es un juego de dos contrincantes, probablemente de origen asiático. Los materiales necesarios para la realización del juego son muy sencillos. No es necesario disponer de un tablero especial; es suficiente con un espacio en el que puedan disponerse 14 orificios o 14 recipientes (según sea el material sobre el que se va a jugar), y 72 pequeñas piedras o cualquier tipo de cuenta de reducido tamaño. La disposición de los orificios debe ser la que muestra la figura 17.

Cada uno de los jugadores es dueño de 7 de los orificios. De ellos, 6 se encuentran alineados y el otro es el denominado kalah, que tiene un



Fig. 17.

papel importante en el juego. La disposición del kalah de cada jugador debe ser fija respecto a la situación de los orificios, tal como muestra la figura 17. En el caso de la figura, el jugador 1 sería dueño de los orificios etiquetados en mayúsculas, mientras que el jugador 2 poseería los etiquetados en letras minúsculas.

Las 72 piedras o cuentas se reparten equitativamente al inicio del juego y cada jugador debe colocar las 36 que le corresponden distribuidas de 6 en 6 en cada uno de sus orificios, quedando el kalah vacío. Así, al comienzo del juego la situación del tablero debe mostrar 6 piedras en cada uno de los orificios A, B, C, D, E, F, a, b, c, d, e y f, mientras que los kalah k1 y k2 no deben contener piedra alguna.

El objetivo del juego es conseguir introducir el mayor número posible de piedras en el kalah, de forma que gana la partida el jugador que consigue tener en su kalah más de la mitad de las piedras que intervienen en el juego, es decir, más de 36 piedras. Cuando tal cosa ocurre, el juego termina, pero también puede darse por terminado el juego cuando todos los orificios de uno de los jugadores, excluido el kalah, queden vacíos (incluso si este hecho se produce cuando no es su turno). Si esto sucede, las piedras que quedan en los orificios del contrario se colocan en su propio kalah y termina el juego, procediéndose al recuento de las piedras para saber cuál de los dos es el ganador.

La forma de realizar un movimiento en el juego del kalah consiste en tomar un número cualquiera de piedras de uno de los orificios propios. A continuación, se van colocando las piedras una por una en sus propios orificios y en los del contrario, recorriendo el tablero de juego en sentido contrario a las agujas del reloj. Por supuesto, entre los orificios propios se cuenta el kalah. El jugador debe colocar exactamente una piedra en cada orificio, ya sea suyo o del contrario, quedando excluido el kalah de su oponente, en el que no dejará ninguna de las piedras que haya tomado.

El siguiente movimiento que ha de realizarse tras el de un jugador está determinado por el lugar en el que este último depositó la última piedra que había tomado. Las situaciones posibles son tres:

— Si el jugador coloca la última piedra en su propio kalah, su turno se mantiene, es decir, puede mover de nuevo.

— Si el jugador coloca la última piedra en uno de sus propios orificios, que estaba vacío y ocurre que el orificio del contrario situado justo en frente (A y a, por ejemplo) tiene al menos una piedra, el jugador debe colocar su piedra y todas las que contenga el orificio del contrario en el kalah de este último, perdiendo su turno de mover, es decir, es ahora el contrario el que puede realizar su movimiento.

— Si el jugador coloca la última piedra en un orificio que no cumpla las condiciones anteriores, no ocurre nada especial, y es el contrario el que mueve a continuación.

La estructura y desarrollo del juego hacen conveniente una implementación computacional que haga uso de algún procedimiento de búsqueda. Distintas realizaciones han optado por la utilización del procedimiento alfa-beta. Para el caso del kalah, la función de evaluación que utiliza el procedimiento se basa en un parámetro k que se denomina «ventaja kalah». Por definición, este valor es, en cada momento, igual a la diferencia entre el número de piedras que contiene el kalah de la máquina y el número de piedras que contiene el kalah del contrincante humano. También puede utilizarse una función de evaluación más compleja, añadiendo al parámetro k una distribución de probabilidad que resulte adecuada al juego.

Los programas ya realizados para este juego han dado prestaciones muy satisfactorias, por lo que no se ha abordado la tarea de la implementación del juego utilizando alguna otra técnica de IA. Por otra parte, la estructura y formulación del juego no parecen prestarse al uso de técnicas más sofisticadas que la búsqueda heurística.

Respecto a los programas que están en estos momentos funcionando, pueden mencionarse los realizados por John McCarthy y alguno de sus estudiantes, y por John Dixon y James R. Slagle. Ambos utilizan el procedimiento alfa-beta, aunque con ligeras variaciones en cuanto al criterio de terminación y la ordenación de los nodos del árbol de búsqueda.

El funcionamiento de estos programas puede considerarse muy bueno, ya que casi siempre derrotan a sus adversarios humanos. Desafortunadamente, estos programas no tienen la posibilidad de aprender a la vista de los resultados y desarrollos de diferentes partidas.

McCarthy demostró con su progama que existe una estrategia vencedora, es decir, una secuencia de movimientos que llevan al jugador a la victoria independientemente de lo que haga el contrario, incluso si éste es el programa. Un contrincante humano puede llegar a descubrir esta secuencia de movimientos mediante un procedimiento de prueba y error, o puede hacerlo más fácilmente haciendo que el programa juegue contra él mismo.



MASTERMIND

Este juego consiste en utilizar la lógica para descifrar una clave de colores determinada anteriormente. El método consiste en hacer una hipótesis, ver los resultados que se obtienen y, en base a esta información, realizar más hipótesis.

La información que se obtiene consiste en el número de colores que coinciden con la clave, haciendo distinción entre los colores que están co-

locados correctamente y los que aun perteneciendo a la clave, no están situados en su lugar correcto.

Una forma de resolver el problema es utilizar todas las informaciones obtenidas hasta el momento y generar, a partir de ellas, las posibles combinaciones de colores que no contradigan ninguna premisa. Una vez hecho esto, se elige una de ellas y se utiliza como nueva hipótesis, y en función de la información obtenida se repite el proceso. La eficacia del programa depende de lo acertada que sea la elección de una nueva clave. El número de jugadas necesarias para descubrir la clave no suele pasar de cinco.

La formalización se puede realizar mediante un Sistema de Producción, en el que:

A) La Base de Hechos está formada por todos los códigos de colores que no contradicen las hipótesis. En principio, la base estará formada por todas las combinaciones posibles. Si se juega con seis colores y el código es de cuatro colores, pudiendo repetirlos, se tiene un total de 4.096 posibilidades. Y en cada momento contiene las combinaciones que no contradicen ninguna información recibida.

La estructura de la Base de Hechos puede ser la siguiente:

$$((c_{11}\ c_{12}\ c_{13}\ c_{14})\ (c_{21}\ c_{22}\ c_{23}\ c_{24}) \ldots\ldots (c_{n1}\ c_{n2}\ c_{n3}\ c_{n4}))$$

donde c_{ij} puede ser:

1. Un color simple. Es decir, uno de los seis colores, está colocado en la posición j en la combinación i, en este caso, c_{ij} es un átomo.
2. Un color compuesto. Una serie de colores pueden estar colocados en la posición i en la combinación j. En este caso, c_{ij} es una lista.
3. Un asterisco, todos los colores pueden estar en la posición i en la combinación j.

Esto permitirá reducir considerablemente el volumen de la base de hechos. Así, la base inicial se puede representar por:

((****))

Una combinación como la siguiente:

((R V) AZ AM B)

se interpreta como si se tuviesen las dos combinaciones siguientes:

((R AZ AM B) (V AZ AM B))

Los seis colores posibles con los que se trabaja son:

Rojo: R Azul: AZ Amarillo: AM Verde: V Blanco: B Negro: N

Las reglas consistirán en formar todas las combinaciones posibles en función de la información obtenida.

Las posibles informaciones son:

— 1 Cero.
— 2 Un muerto.
— 3 Un herido.
— 4 Un muerto y un herido.
— 5 Un muerto y dos heridos.
— 6 Un muerto y tres heridos.
— 7 Dos muertos.
— 8 Dos heridos.
— 9 Dos muertos y un herido.
— 10 Dos muertos y dos heridos.
— 11 Tres muertos.
— 12 Tres heridos.
— 13 Cuatros heridos.
— 14 Cuatro muertos.

Cada una de estas informaciones va a dar lugar a un número finito de posibles combinaciones de colores que restringen cada vez más el número de soluciones posibles.

Por convención se utiliza la variable A para representar el color que aparece en el primer lugar de la hipótesis que se está tratando, B para el correspondiente al segundo lugar, C para el tercero y D para el cuarto. En cada caso estos colores tienen un valor determinado. Además, se utiliza la variable $, que contendrá una lista con los colores que no aparecen en la hipótesis.

La información número 1 dará lugar a que se supriman en la base de datos todas las combinaciones que contengan uno de los colores de la hipótesis en una posición simple, que desaparezca el color en todas las combinaciones que lo posean en una posición compuesta y en caso de que una posición posea un asterisco, se transformará en una posición compuesta con los colores que no aparecen en la hipótesis. Esto se puede representar, como regla, de la siguiente forma:

SI 1 ENTONCES ($$$$)

Si se obtiene la información 2, las posibles combinaciones son:

(A$$$) ($B$$) ($$C$) ($$$D)

que darán lugar a cuatro reglas:

Si 2 ENTONCES (A$$$)
Si 2 ENTONCES ($B$$)
Si 2 ENTONCES ($$C$)
Si 2 ENTONCES ($$$D)

que se puede compactar de la siguiente forma:

Si 2 ENTONCES (A$$$) ($B$$) ($$C$) ($$$D)

La información 3 da lugar a la siguiente regla:

Si 3 ENTONCES ((A$) A (A$) (A$)
((A$) (A$) A (A$))
((A$) (A$) (A$) A)
(B (B$) (B$) (B$))
((B$) (B$) B (B$))
((B$) (B$) (B$) B)
(C (C$) (C$) (C$))
((C$) C (C$) (C$))
((C$) (C$) (C$) C)
(D (D$) (D$) (D$))
((D$) D (D$) (D$))
((D$) (D$) D (D$))

Para el caso 4 las combinaciones posibles darían lugar a la siguiente regla:

Si 4 ENTONCES (A (A$) (AB$) (AB$))
(A (AC$) (A$) (AC$))
(A (AD$) (AD$) (A$))
((BC$) B (C$) (BC$))
((BD$) B (BD$) (B$))
((CD$) (CD$) C (C$))
((B$) B (AB$) (AB$))
((C$) (AC$) C (AC$))
((D$) (AD$) (AD$) D)
((BC$) (C$) C (BC$))
((BD$) (D$) (BD$) D)
((CD$) (CD$) (D$) D)

A partir de aquí, las reglas se complican debido a que aumenta el número de combinaciones concretas posibles, pero en comparación, restringe más el campo de búsqueda.

Si aparece la información número catorce, entonces se ha encontrado la combinación correcta:

SI 14 ENTONCES (A B C D)

La aplicación de una regla consistirá en tomar cada una de las partes izquierdas y compararla con cada combinación de la base de datos. La nueva base de hechos se crea con las combinaciones que cumplen las condiciones.

La estrategia a seguir consiste en ejecutar todas las reglas posibles formando el conjunto de combinaciones que no contradicen la información

recibida. Para ello se comparan todas las combinaciones que se crean con las reglas con las que ya existen en la base de hechos. La nueva base de hechos se obtiene de la intersección de ambos conjuntos. La elección de la siguiente hipótesis se realiza de la siguiente forma: se realiza una estadística que refleje la probabilidad de que un color aparezca en una determinada posición, para cada color y cada posición. Y se elige para nueva hipótesis los colores que tengan más probabilidades de aparecer en cada posición.



EL NIM

Este juego, a pesar de ser muy sencillo, posee muchas variantes de práctica. La más común es la que va a explicarse en este apartado. Las reglas son simples: los jugadores extraen alternativamente de una fila de objetos el número de objetos que deseen, dentro del margen comprendido entre 1 y la mitad del total de objetos que queden en la fila. Por ejemplo, si en un momento dado la fila está formada por 21 objetos, podrá retirarse un número de objetos comprendido entre 1 y 10. Ganará aquel jugador que retire el último objeto de la fila.

La estrategia ganadora de este juego es muy conocida: se ha de intentar dejar siempre al oponente un número de objetos que pertenezca a la serie 2^{n-1}. Es decir, los números 1, 3, 7, 15, 31, 63, ... Estos números reciben también el nombre de posiciones «seguras». Se denominan así puesto que, con independencia de las posibles jugadas del contrincante, estas posiciones forman un camino que conduce a la victoria dentro del árbol de búsqueda. Se puede comprobar que si se desarrolla un árbol «Y-O» de este problema, se obtiene un camino seguro a la meta (se gana al contrincante), al seguir esta estrategia.

Si se estudia esta estrategia pueden deducirse una cuantas conclusiones. Si al principio del juego la fila está constituida por un número de objetos perteneciente a dicha serie, entonces el ganador será el que juegue en segundo lugar. Esto se cumplirá siempre que éste siga la estrategia descrita. La explicación es elemental: si hay 2^{n-1} objetos al principio, sólo podrá quitar un número de objetos comprendido entre 1 y $(2^{n-1})/2$. Pero el siguiente número de la serie al 2^{n-1} es el $2^{n-1}-1$. Este número no puede alcanzarse si sólo podemos quitar un número entre 1 y $(2^{n-1})/2$. Por tanto, el primero en dejar $2^{n-1}-1$ objetos en la fila será el segundo jugador. Si se parte de la base de que juega siguiendo la estrategia descrita, entonces ganará irremisiblemente.

De igual forma se puede llegar a la conclusión de que si el número de partida no pertenece a dicha serie, el ganador será el primer jugador, si sigue la estrategia. De todo esto se extrae la conclusión de que si los dos ju-

gadores siguen la estrategia ganadora, el ganador se conoce desde el principio del juego. Por tanto, es un juego, en cierto modo, determinista.

Todo este proceso es muy fácil de implementar mediante cualquier programa de computador. Ahora bien, es muy sencillo también construir un Sistema de Producción en base a la estrategia. La ventaja de un Sistema de Producción en este caso es que está autodocumentado, como todos ellos. Por tanto, es fácil seguir su funcionamiento. La Base de Hechos estará formada por el número de objetos que quedan en la fila. Al principio, la Base contendrá el número inicial de objetos con que se desee comenzar.

La Base de Reglas estará formada por sólo dos reglas:

SI número $\in$ serie 2^{n-1} ENTONCES Quitar (1)
SI número $\notin$ ENTONCES Quitar (Calcular-Número)

donde,

— el procedimiento Quitar (n), quita n objetos de la fila. Es decir, cambia el valor del único elemento de la Base de Hechos de la siguiente forma:

número = número − n

— la función Calcular-Número, calcula el número perteneciente a la serie, inmediatamente inferior al número de la Base de Hechos.

La Estrategia de Control pedirá las jugadas del contrincante, preguntándole el número de objetos que quiere retirar de la fila. Con ese número llamará al procedimiento Quitar y, por último, disparará la regla que se pueda ejecutar, comenzando el ciclo otra vez. Puede observarse que este Sistema de Producción no podrá tener nunca conjunto conflicto y además resulta demasiado elemental.

La importancia de este juego consiste en comprobar que existen estrategias ganadoras que ya desde el principio del juego dirigen el sistema hacia la victoria, dependiendo de la situación inicial.



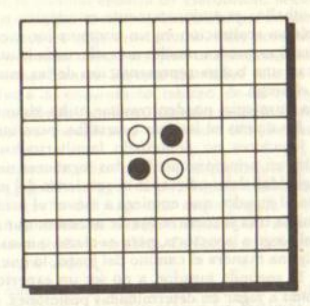
«OTHELLO»

El juego del «othello» tiene origen japonés, y se ha extendido profusamente por todos los países del mundo gracias a su comercialización y a su sencillez. Goza de la fama de ser un juego extremadamente sencillo de conocer, pero difícil de dominar en su totalidad. La leyenda atribuye su dominio a los monjes que dedican gran parte de su vida a la meditación, y que, al parecer, utilizan el juego como parte de su aprendizaje.

Los materiales necesarios para el juego son un tablero con 64 cuadros sin distinción de colores, y 64 fichas circulares planas con un color diferente en cada cara, una de ellas blanca y la otra negra.

Los jugadores eligen un color (blanco o negro) y mantienen ese color

durante todo el juego. Antes de iniciarse la partida deben colocarse cuatro fichas sobre el tablero de la siguiente forma:



A partir de esta situación el primer movimiento está reservado para el jugador que utiliza las fichas de color negro. Un movimiento consiste en la colocación de una ficha al lado de otra del oponente, ya sea en dirección horizontal, vertical o diagonal. Esta ficha tiene la misión de atrapar una o varias de las fichas del jugador contrario. Quedan atrapadas todas las fichas de distinto color que se encuentren situadas entre dos fichas del jugador que mueve. Las fichas capturadas se dan la vuelta y quedan del color del jugador que las atrapó pasando así a ser de su propiedad. El número de fichas que pueden ser capturadas en una jugada en cualquier dirección o en varias direcciones sólo se encuentra limitado por el número de fichas que en el momento estén sobre el tablero.

Los jugadores realizan sus colocaciones de fichas en turno alternativo. Si en uno de sus turnos uno de los jugadores no tiene posibilidad de capturar una ficha de su oponente, perderá el turno y no podrá jugar hasta que pueda colocar una ficha que capture al menos una ficha de su rival. Mientras uno de los jugadores no puede poner, el otro continúa jugando, dando la vuelta a todas las fichas que le sea posible.

El juego termina cuando sucede alguna de las siguientes cosas:

— El tablero se ha llenado de fichas.

— A uno de los jugadores no le queda ninguna ficha de su color sobre el tablero.

— Ninguno de los jugadores puede colocar una ficha que atrape otra del oponente.

Una vez terminando el juego, se cuenta el número de fichas de cada color y gana el jugador que haya acumulado más fichas de su color.

El juego del «othello» es eminentemente estratégico y, aunque es perfectamente factible su realización en un computador mediante una búsqueda de cualquier tipo, parece mucho más adecuada la utilización de métodos que permitan una buena representación de las estrategias que deben aplicarse en el juego.

En cuanto a la estrategia, pueden resultar útiles algunas ideas que no son, en absoluto, las únicas ni las más acertadas, pero que pueden guiar las acciones de jugadores no demasiado familiarizados con el juego.

Hay que señalar en principio que los dos jugadores no se encuentran exactamente en igualdad de condiciones al comienzo del juego. Existe una ligera ventaja para el jugador que empieza a mover el primero. No puede asegurarse que exista una secuencia fija de acciones que conduzca al jugador que inició el juego a la victoria, pero es cierto que su posición le permite marcar de alguna manera el camino del juego, lo que le concede una evidente ventaja. El segundo jugador, a no ser un experto en la materia, irá siempre obligado a jugar en determinadas posiciones, que es de esperar no sean las que más favorezcan su juego. Suele recomendarse la alternancia del color de las fichas entre los jugadores, de forma que el turno de salida se alterne de la misma manera, y no corresponda la ventaja siempre al mismo jugador.

Una buena estrategia de juego consiste en hacerse dueño lo antes posible de los rincones del tablero, es decir, de los cuadros situados en las esquinas. Estos puntos son posiciones especialmente fuertes para el jugador que se apodera de ellos, ya que la ficha colocada en una esquina no puede ser nunca atrapada por otra ficha del contrario. De igual forma, también las posiciones de los bordes del tablero tienen importancia, ya que las fichas situadas sobre ellas sólo presentan dos direcciones de ataque, por lo que pueden ser vigiladas con más facilidad.

En general, puede decirse que un jugador debe tender a realizar un bloqueo con fichas de su color, a ser posible en un rincón o en un borde, lugares en los que resulta más sólido, y en la mayoría de las ocasiones conduce a la victoria. También cabe señalar que los buenos jugadores no recomiendan siempre la jugada que atrape el mayor número de fichas, sino aquella que tiende a una situación de mayor dominio posicional sobre el tablero.

Por lo que se refiere a la programación del juego propiamente dicha, hay que indicar que puede realizarse utilizando el procedimiento alfa-beta sin demasiados problemas. El tablero puede representarse mediante una matriz de 10 por 10, aunque sus dimensiones son 8 por 8. Este aumento en el número de casillas se utiliza para facilitar la verificación de las jugadas. Así, una jugada será válida si permite capturar una o varias fichas contrarias y si la posición se encuentra en la submatriz de 8 por 8 que representa el tablero real.

El cálculo de la función estática de evaluación, necesaria para el procedimiento, se realiza por medio de una tabla que contiene el valor estratégico que se concede a cada movimiento. Para conceder un valor estratégico a cada posición se utiliza también una matriz auxiliar que contiene los valores correspondientes. En esta matriz los valores más altos son los que corresponden a las esquinas del tablero, de las que ya se ha comentado su importancia. Evidentemente, los valores de las posiciones de la matriz auxiliar pueden modificarse con cada una de las jugadas que se realizan.

Es posible mejorar la eficacia de la búsqueda si se examina en cada situación la mejor jugada en lugar de una jugada cualquiera de las posibles. Aunque en el desarrollo del juego esta mejor jugada se desconoce, puede aún realizarse una considerable mejora estudiando, en primer lugar, las jugadas aparentemente más favorables. Es decir, se deberían estudiar las jugadas en el siguiente orden:

— Jugadas que llevan consigo la pérdida de la esquina.
— Jugadas en posiciones adyacentes a las esquinas.
— Jugadas peligrosas en las orillas de una zona tomada por el adversario.
— Jugadas en la cruz central del tablero.
— Jugadas favorables en las orillas.

La ordenación de estas jugadas se realiza mediante el estudio del juego, de acuerdo con las estrategias que ya se han mencionado.



JUEGO DEL PUENTEO

El juego del puenteo, también denominado del camino, constituye un buen ejemplo de juego posicional, es decir, cuyo desarrollo depende casi totalmente de las posiciones, tanto absolutas como relativas, de los distintos elementos sobre el tablero de juego. Otros ejemplos de juegos de este tipo son el «tic-tac-toe», «hex», «qubic», etc.

En este juego, el tablero es un rectángulo cubierto con puntos blancos y negros alternos por filas como muestra la figura 18.

Como puede verse, los bordes opuestos son del mismo color. En el juego intervienen dos jugadores. Cada uno de ellos debe elegir uno de los dos colores, que será el que mantenga durante el desarrollo completo de una partida. El juego consiste en unir, por turnos alternativos, dos de los puntos contiguos de un color mediante una línea. Un jugador sólo puede utilizar los puntos del color que ha elegido antes del comienzo de la partida.

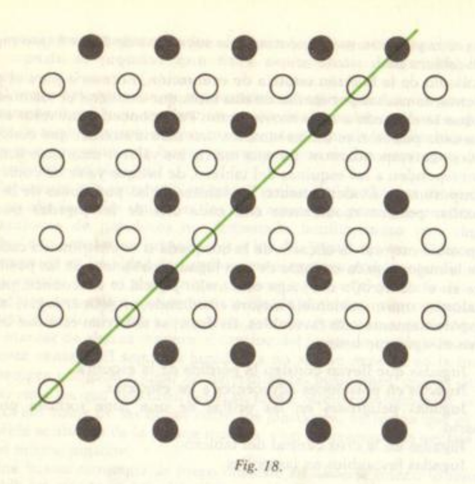


Fig. 18.

El objetivo de cada jugador es conseguir llegar desde el borde de partida hasta el borde opuesto mediante una cadena de trazos sin solución de continuidad, es decir, mediante una línea continua. La única restricción existente es la imposibilidad de cruzar en ningún caso una línea dibujada por el jugador contrario.

Suele considerarse, por tanto, un borde de cada color como origen y el otro como meta, de forma que los jugadores tratan de construir un «puente» desde uno de los puntos del borde origen hasta uno de los del borde meta.

Puede demostrarse sin demasiada dificultad que en este juego existe una estrategia ganadora para el jugador que realiza el primer movimiento. Esta interesante característica es común a la mayoría de los juegos posicionales.

La resolución de este juego mediante un programa de computador no parece seguir el camino de los procedimientos de búsqueda clásicos en juegos. La aparente simplicidad de las estrategias que pueden utilizarse hacen más adecuada la realización de programas que tengan presente la estructura CONDICION → ACCION, típica de los Sistemas de Producción.

Una forma de tratamiento del juego, descubierta por Oliver Gross, utiliza una estrategia bastante simple. Para su explicación considérese el tablero dividido por la diagonal AB como muestra la figura 19. Supóngase que el programa es el jugador que utiliza los puntos negros. Dependiendo de la elección del oponente, el programa debe replicar con las siguientes acciones:

— Si el oponente une uno de sus círculos con una línea horizontal, la respuesta debe ser la que muestra la figura 20.

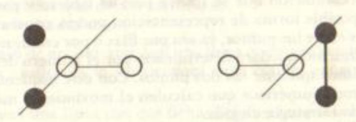


Fig. 20.

— Si el oponente une dos de sus puntos que se encuentran por encima de la diagonal, entonces replicar de la forma que muestra la figura 21.

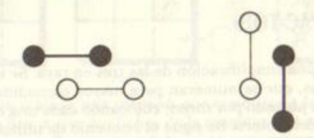


Fig. 21.

— Si el oponente une dos de sus puntos situados por debajo de la diagonal, la respuesta debe ser la expresada en la figura 22.

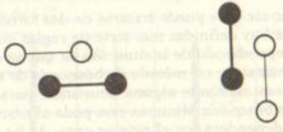


Fig. 22.

Si el programa comienza el juego y sigue esta estrategia, su victoria está garantizada. Su primer movimiento debe unir los puntos que se muestran en la figura 23.

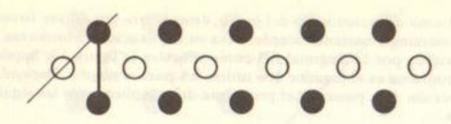


Fig. 23.

La forma específica de programar esta estrategia depende en gran medida de la representación que se utilice para el tablero y para los movimientos. Una posible forma de representación podría consistir en numerar los espacios entre los puntos, ya sea por filas o por columnas, de forma que cada movimiento quedará determinado por el número del hueco que ha cruzado la línea que une los dos puntos. Con este esquema es posible utilizar algoritmos numéricos que calculen el movimiento más ventajoso en función de la estrategia elegida.

Otra posibilidad es representar la estrategia en forma de una tabla de valores que tipifique cada réplica a un movimiento dado. El programa, entonces, debe guiar, sus acciones mediante búsqueda en las tablas.



«TIC-TAC-TOE»

Este juego es una simplificación de las tres en raya. Se tiene un tablero de 3 por 3 casillas, que se numeran para mayor comodidad. Hay dos adversarios que van jugando por turno, colocando cada una de sus fichas en una casilla libre del tablero. Se sigue el convenio de utilizar las cruces 'x' para el jugador y las caras '0' para el computador.

El juego termina cuando uno de los jugadores ha completado con sus fichas una fila, columna o diagonal de tres casillas, resultando ganador. O bien cuando se ha cubierto con las fichas todo el tablero, terminando el juego en tablas.

La partida de «tic-tac-toe» puede tratarse de dos formas. La primera es un programa en que hay definidas una serie de reglas que realizan uno u otro movimiento, dependiendo de la situación en que se encuentra la partida. La segunda forma sigue un método de búsqueda de situaciones y evaluación de las mismas mediante alguna heurística, para elegir la mejor. Puede aplicarse una búsqueda Minimax con poda alfa-beta. La bondad de la implementación dependerá, en el primer caso, de las reglas y la estrategia de control, y en el segundo, de la función heurística.

En la primera forma de implementación la elección de la casilla a jugar se realiza mediante un Sistema de Producción. La Base de Datos está formada por dos listas (L1 para la persona y L2 para la máquina) que contienen la casillas ocupadas por cada uno de los jugadores. Por otro lado, existen una serie de cláusulas definidas al principio del programa que permitirán disponer de un conjunto de propiedades de las casillas del tablero necesarias en algún momento determinado. Estas propiedades pueden ser las casillas que hay en el tablero, las casillas situadas en las esquinas y las centrales, así como propiedades de la línea.

Las reglas constan de una parte de condición y otra de accion, que consiste simplemente en seleccionar la casilla que corresponde jugar, y que se incuye en la lista L2 de la máquina. La parte de condición son el conjunto de predicados que deben verificarse.

Las reglas incluidas son:

a) En caso de encontrarse con una línea que tiene dos fichas de la máquina y una casilla vacía, se ocupará esta casilla, ganando la partida. Ver fig. 24.

b) Se tiene una línea con dos fichas del jugador y una casilla vacía, se ocupa ésta para evitar que gane el jugador. Ver fig. 25.

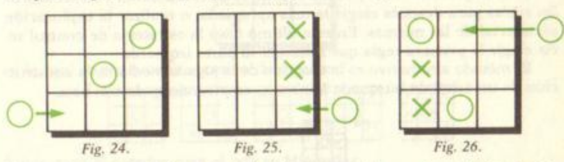


Fig. 24. *Fig. 25.* *Fig. 26.*

c) Se intenta realizar una ofensiva que conduzca a la victoria, al conseguir tener dos líneas con dos fichas de la máquina en cada línea y las restantes casillas vacías. Para conseguirlo, si se encuentran dos líneas cada una con una ficha de la máquina y el resto vacías, se ocupa la casilla intersección de las dos líneas. Ver fig. 26.

d) Se intenta responder satisfactoriamente a una situación del mismo tipo que en c). En este caso, se debe ocupar una casilla central de una de las dos líneas. Ver fig. 27.

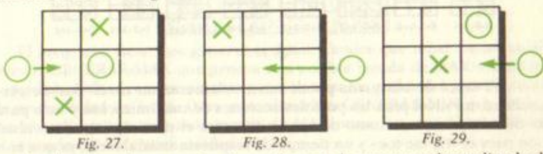


Fig. 27. *Fig. 28.* *Fig. 29.*

Las dos situaciones son iguales, sólo varia la respuesta dependiendo de quién lleve la iniciativa.

e) Se ocupa el centro del tablero en caso de estar vacío debido a su importancia. Ver fig. 28.

f) Cuando se tiene una línea que contiene sólo una ficha de la máquina, se coloca otra ficha para tener dos en raya, llevando así la iniciativa. Ver fig. 29.

g) Se ocupa cualquier casilla que esté vacía. Ver fig. 30.

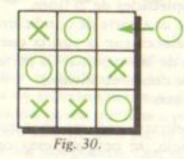


Fig. 30.

En caso de no jugar bien bastaría con cambiar las reglas anteriores por otras mejores.

En la implementación pueden realizarse una restricción y un filtrado de las reglas para después elegir la más apropiada, o realizar la exploración secuencial de las mismas. En este último caso la estrategia de control sería elegir la primera regla que satisfaga su parte izquierda.

El método alternativo es la elección de la jugada mediante la construcción de un árbol de búsqueda Minimax, empleando poda alfa-beta.

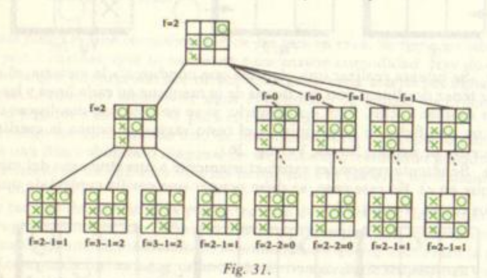


Fig. 31.

El árbol de búsqueda puede tener solamente un nivel 'dos' de profundidad, un nivel para las posibles acciones de maximización y otro para las de minimizacion, como compromiso entre el nivel óptimo de evaluación para el «tic-tac-toe» y un tiempo de respuesta satisfactorio, ya que con una profundidad de 'cuatro' se podría afrontar cualquier situación sin posibilidad de sorpresas.

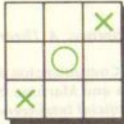
El árbol sólo tendrá un nivel 'MAX' y otro 'MIN'; como consecuencia, no va a realizarse ningún corte por debajo de ningún nodo 'MAX'. El algoritmo alfa-beta se reducirá a un algoritmo Minimax con cortes alfa.

La función de evaluación que se utiliza para una posición P es:

f(P) = Σ filas que todavía están libres para MAXimización.
Σ filas que todavía están libres para MINimización.

El funcionamiento para una situación cualquiera sería el especificado en la figura 24.

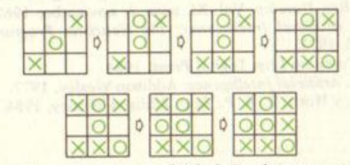
Hay una situación que no se resuelve bien; esto se debe a la falta de profundidad del árbol para prever las situaciones que se van a presentar; con un nivel 'cuatro' de profundidad no hay ningún problema. La situación es la siguiente:



en la cual se produce la sucesión de jugadas:

donde gana el jugador para el que se MINimiza.

Cuando la respuesta correcta debería haber sido:



El programa tiene que generar el árbol. Tendrá que tener un procedimiento GENERAR-MAX que genera una posible jugada de MAXimización y que llamándose recursivamente genera todos los nodos del primer nivel de profundidad de árbol. Otro procedimiento GENERAR-MIN que se llama recursivamente por cada jugada de MAXimización, con él se obtiene el segundo nivel de profundidad.

La elección de una jugada corresponde a la generación y evaluación simultánea del árbol de búsqueda.

BIBLIOGRAFIA

Banerji, R. B.: *Artificial Intelligence: A Theoretical Approach.* North-Holland, 1980.
Collins, D.: *Search Strategies.* Comunicación Personal.
Frey, P. W.: *Chess Skill in Man and Machine.* Springer-Verlag.
Nilsson, N. J.: *Principles of Artificial Intelligence.* Tioga, 1980.
Pazos, J.: *Luciano: An Expert System to Play Dominoes.* WorkShop IEEE, 1985.
Pazos, J.: *Inteligencia Artificial: Programación Heurística.* Paraninfo (en prensa).
Pearl, J.; *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984.
Rich, E.: *Artificial Intelligence.* McGraw-Hill, 1983.
Samuel, A.: *Some Studies in Machine Learning Using the Game of Checkers, IBM J. Res, Develop.* Vol. XI, núm. 6, noviembre 1967.
Slagle, J. R.: *Artificial Intelligence: the Heuristic Program Approach.* McGraw-Hill, 1971.
Steele, G. L.: *Common Lisp.* Digital Press, 1984.
Winston, P. H.: *Artificial Intelligence:* Addison-Wesley, 1977.
Winston, P. H., y Horn, B. K. P.: *Lisp.* Addison-Wesley, 1984.